LY20-2228-1

Systems

IBM System/370 Special Real Time Operating System Programming RPQ Z06751 Systems Logic Manual

Program Number 5799-AHE

This publication describes the internal logic and method of operation of the Special Real Time Operating System. The purpose of this publication is to provide information for systems analysts, programmers, systems engineers, and maintenance personnel to facilitate making modifications, diagnosing error situations, and performing maintenance.

IBM

Second Edition (August 1983)

This is a reprint of LY20-2228-0 incorporating changes released in Technical Newsletter LN20-3622 (dated 31 August 1976). This edition applies to Version 1, Modification 1 of the Special Real Time Operating System PRPQ (Programming Request for Price Quotation) number 5799-AHE, Version 1, Modification 1, and to all subsequent versions and modifications until otherwise indicated in new editions or Technical Newsletters.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Changes are periodically made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the latest *System/370 Bibliography* (GA22-6822) for the editions that are applicable and current.

The PRPQ described in this manual, and all licensed materials available for it, are provided by IBM on a special quotation basis only, under the terms of the License Agreement for IBM Program Products. Your local IBM branch office can advise you regarding the special quotation and ordering procedures.

A form for readers' comments has been provided at the back of this publication. If the form has been removed, address comments to IBM Corporation, Dept. 824, 1133 Westchester Ave., White Plains, New York 10604. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

PREFACE

Scope and Objectives

This publication describes the internal logic and method of operation of the Special Real Time Operating System. The purpose of the publication is to provide information to systems analysts, programmers, system engineers, and maintenance personnel to facilitate making modifications, diagnosing error situations, and performing maintenance work.

All of the general functions and services provided by the Special Real Time Operating System are described in the DOM, as well as the details and requirements for installing, operating, and customizing this PRPQ. However, very little emphasis is given to individual programs that comprise these services or to the overall organization of these programs. The SLM contains the detailed material pertaining to the design and coding of the Special Real Time Operating System. While the DOM concentrates on the services and how to use them, the SLM concentrates on the load modules and how they perform their functions. The SLM is not intended to replace or duplicate any information found in the DOM; it supplements the DOM with information for diagnosing error situations, or making modifications.

This publication contains three sections:

1. Section 1. Introduction

   - summarizes general information about the Special Real Time Operating System
   - describes the relationship between the functional operations-oriented information contained in the Special Real Time Operating System Description and Operations Manual (SH20-1773) and the detailed program-oriented information contained in this manual
   - describes the relationships between the various sections and appendixes of this manual

2. Section 2. Logic description

   - summarizes the organization of the various functional areas that compose the Special Real Time Operating System
   - defines the format of the diagram and the symbols used to describe the individual programs that comprise the various functional areas
   - provides a visual description of the major logical processes for the individual programs through use of Hierarchy Input Process Output (HIPO) diagrams

3. Section 3. Program Organization

   - o provides a detailed listing of the organization of the individual programs and defines the major processes of those programs through the use of Program Design Language (PDL)

This publication contains four appendixes:

1. Appendix A. Directory

   - o contains cross-references of the Special Real Time Operating System CSECT names to the source members that comprise that CSECT
   - o contains cross-references from CSECT name to the appropriate HIPO and PDL information
   - o contains cross-references from Special Real Time Operating System macro names to CSECT name
   - o contains cross-references from CSECT name to the functional area

2. Appendix B. Storage Allocation

   - o contains information concerning the amount of storage required to execute the Special Real Time Operating System

3. Appendix C. Data Areas

   - o contains charts describing the relationships between various Special Real Time Operating System control blocks and data areas
   - o contains detailed description of each Special Real Time Operating System control block used by multiple modules

4. Appendix D. Internal Macros - Contains Macros used internal to the Special Real Time OPerating System

The page numbering structure of this manual is designed so that the first digit is the section and the remaining digits are the page numbers. For example, 2-34 means you are in Section 2 on page 34.

The references within the logic diagrams are references to other figure numbers _not_ page numbers.

The reader must have a general knowledge of the concepts of Program Design Language and must understand the concepts and techniques involved in using HIPO function. Section 1 Introduction, can help in using this manual effectively.

In addition, the reader must have a thorough understanding of the concepts and a knowledge of the terminology used in OS/VS1.

PREREQUISITE PUBLICATIONS

The reader should be familiar with the concepts presented in the following publications:

Special Real Time Operating System Description and Operations Manual (SH19-0080)

IBM System/370 Principles of Operation (GA22-7000)

OS/VS1 Planning and Use Guide (GC24-5090)

IBM System/370 System Summary (GA25-7001)

RELATED PUBLICATIONS

OS/VS1 Debugging Guide (GC24-5093)

OS/VS1 Supervisor Logic (SY24-5155)

The Special Real Time Operating System PRPQ is a group of programs that augments the services of OS/VS1 to support realtime applications. Additional services provide for lower supervisor overhead; new capabilities; and increased flexibility in the areas of task management, time management, data base, message handling, duplicate data set support, data recordings and playback, failover/restart, and other supplementary services.

The services provided by OS/VS1 are still available to a program or system of programs utilizing the Special Real Time Operating System. However, in some cases, the Special Real Time Operating System may act as an interface between OS/VS1 and user programs.

The Special Real Time Operating System is designed to enhance areas that are critical to a realtime operation and to provide a stable operating environment which will minimize the impact of an abnormally terminating program.

SYSTEM ENVIRONMENT

The Special Real Time Operating System executes as an application program under the control of OS/VS1. There are two distinct modes of operation, either an online job step which executes in conjunction with user programs and/or other program products in a realtime environment, or an offline job step which creates and/or modifies tables, data sets, etc., that are essential for the proper execution of the online job step.

The online job step includes supervisor call instruction (SVC) and contains non-SVC routines which may attain supervisor state and/or supervisor protect key while executing as application load modules. Any other program products and user programs are executed as subtasks to the online job step task. In this environment, all the Special Real Time Operating System services described in the Description and Operations Manual (DOM) are available to the user programs, Special Real Time Operating System routines, and/or other program products.

The offline job step (e.g., offline utility) executes as a separate independent job step from the online (or realtime) job step and, in this environment, the Special Real Time Operating System services, as such, are not available.

SPECIAL REAL TIME OPERATING SYSTEM OVERVIEW

The Special Real Time Operating System is separated into two modes of operation: online excution and offline execution. Each mode of operation is composed of one or more functional areas as shown in Figure 2-4.

Note: When referring to an overview chart, the numbers following the description (usually found in the lower right corner of the boxes), refer to either the figure number of another overview chart or the figure number of the detailed HIPO diagram that describes the routine that is given control.

Each of the functional areas that comprises the Special Real Time Operating System is represented in this section by a brief narrative of that function, an overview chart, where applicable, followed by detailed HIPO charts of the modules involved.

ONLINE EXECUTION

Online execution of the Special Real Time Operating System is initiated through standard OS/VS1 Job Control Language (JCL) statements with the EXEC card specifying PGM=DPPINIT. The JCL defines to the Special Real Time Operating System the data sets which have been created by the offline utility and the Special Real Time Operating System SYSGEN procedures (described in the Description and Operations Manual).

The JCL also defines the devices which are to be used by the online routines. The module DPPINIT is responsible for initializing most of the functional areas for online execution.

Once the basic initialization has been completed, the Special Real Time Operating System performs meaningful processing only when its services are requested, either by user programs executing user macro calls in a realtime environment or by user interfaces such as Input Message Processing commands and/or PATCH statements in the SYSINIT input stream. Figure 2-5 shows the relationships between the user macros and the functional areas. The number following the macro name is the figure number of the HIPO diagram that describes the module that receives control in response to a particular macro call.

Figure 2-6 shows the input message processing operator commands that are recognized by the Special Real Time Operating System. It also shows the entry point names that can be specified on a PATCH statement in the SYSINIT input stream that result in processing by the Special Real Time Operating System.

During normal execution, or as a result of a user request, the Special Real Time Operating System may execute one or more internal macro calls, as well as the user macro calls. Use of these internal macro calls is restricted by the Special Real Time Operating System because they may be used to obtain supervisor state, page fixing, etc. which, without strict controls, could jeopardize the performance or the integrity of the operating system. Figure 2-7 shows the internal macros used by the Special Real Time Operating System. Appendix D contains a list of these macros and their calling sequences.

OFFLINE EXECUTION

The offline functions of the Special Real Time Operating System are executed through the use of standard OS/VS1 Job Control Language (JCL). These functions are the offline utility, data base BDAM data set compress, playback of recorded data, and stage I of the system generation procedure.

HOW TO READ HIPO DIAGRAMS

The HIPO diagrams illustrate the functions performed by the Special Real Time Operating System. Each major functional area has a set of diagrams. The first figure in each set is a visual table of contents for that functional area.

The HIPO diagrams are read left to right, top to bottom, and illustrate the input, the processing steps, and the output for each function performed. The input to the function appears on the left and the output of the function appears on the right. The processing is divided into a series of steps. If further explanation of a processing step is needed, that step is numbered and the explanation appears in the Extended Description for that diagram. The Extended Description also contains segment names, so that the reader can refer to the proper PDL segment or pertinent code in the program listing.

Arrows are used to signify data movement, data reference, and processing flow. The arrow conventions are shown in Figure 2-1. Other conventions used in the HIPO diagrams are illustrated in Figures 2-2 and 2-3.

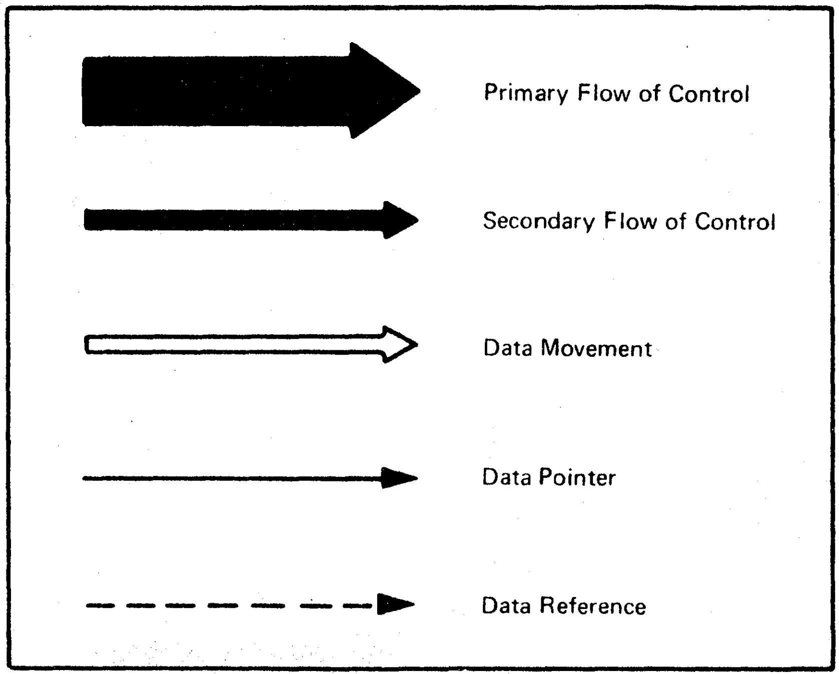
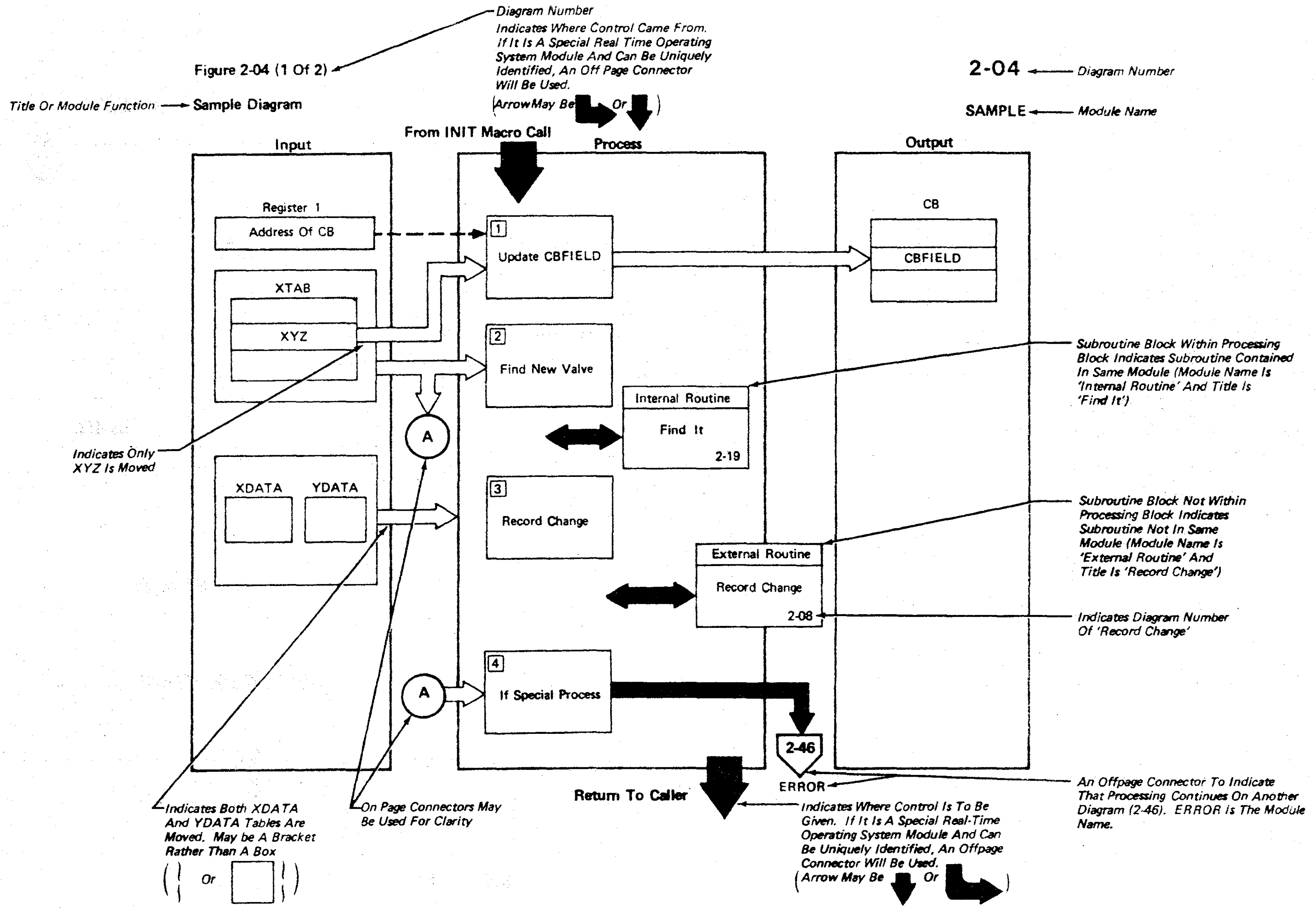


Figure 2-1. HIPO Arrow Conventions



Title Or Module Function

Diagram Number

Indicates Where Control Came From. If It Is A Special Real Time Operating System Module And Can Be Uniquely Identified, An Off Page Connector Will Be Used. (Arrow May Be Or)

Diagram Number

Module Name

Subroutine Block Within Processing Block Indicates Subroutine Contained In Same Module (Module Name Is 'Internal Routine' And Title Is 'Find It')

Subroutine Block Not Within Processing Block Indicates Subroutine Not In Same Module (Module Name Is 'External Routine' And Title Is 'Record Change')

Indicates Diagram Number Of 'Record Change'

Indicates Only XYZ Is Moved

Indicates Both XDATA And YDATA Tables Are Moved. May be A Bracket Rather Than A Box

On Page Connectors May Be Used For Clarity

Indicates Where Control Is To Be Given. If It Is A Special Real-Time Operating System Module And Can Be Uniquely Identified, An Offpage Connector Will Be Used. (Arrow May Be Or)

An Offpage Connector To Indicate That Processing Continues On Another Diagram (2-46). ERROR Is The Module Name.

Figure 2-2 · Sample HIPO Diagram

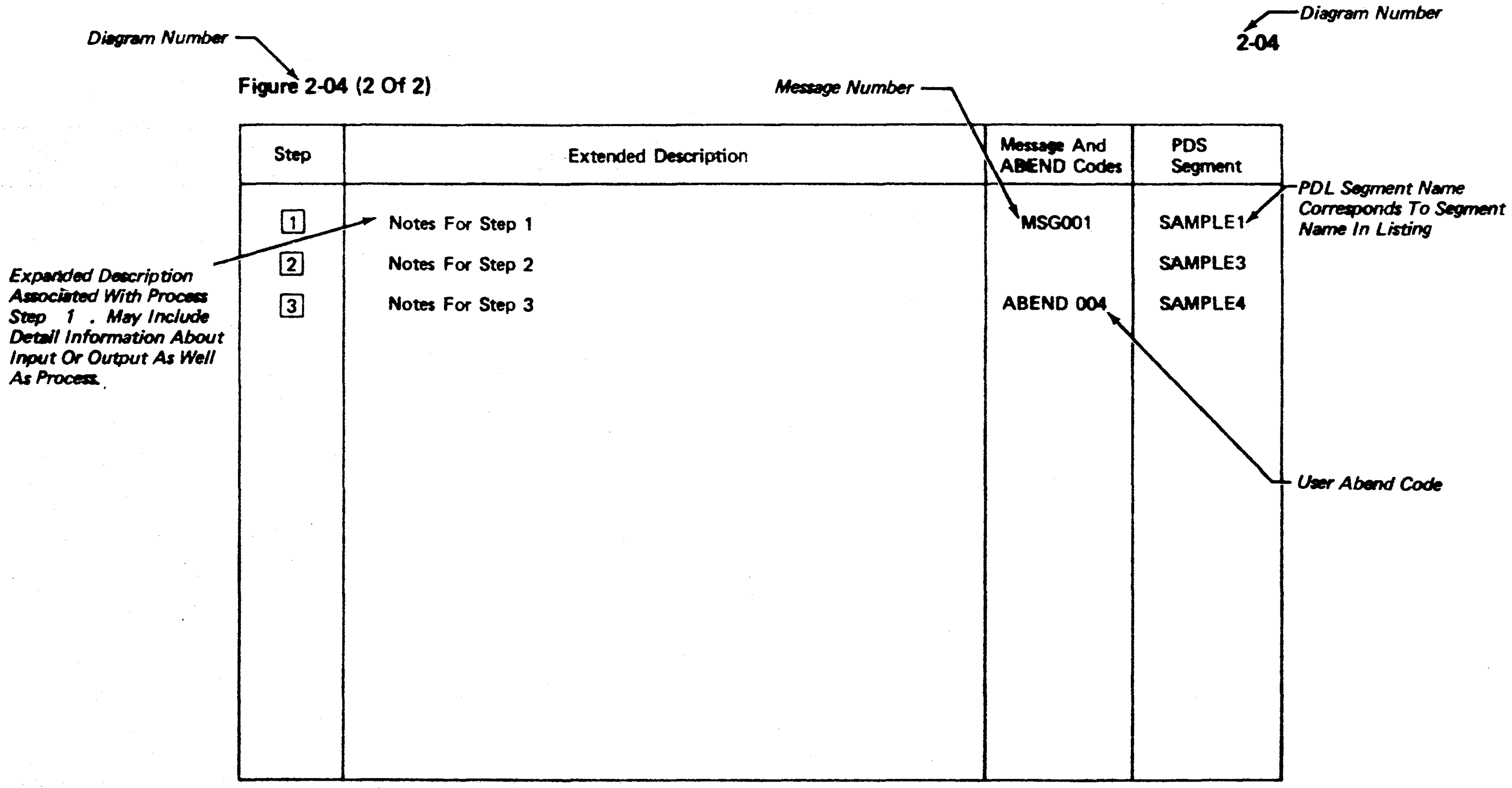


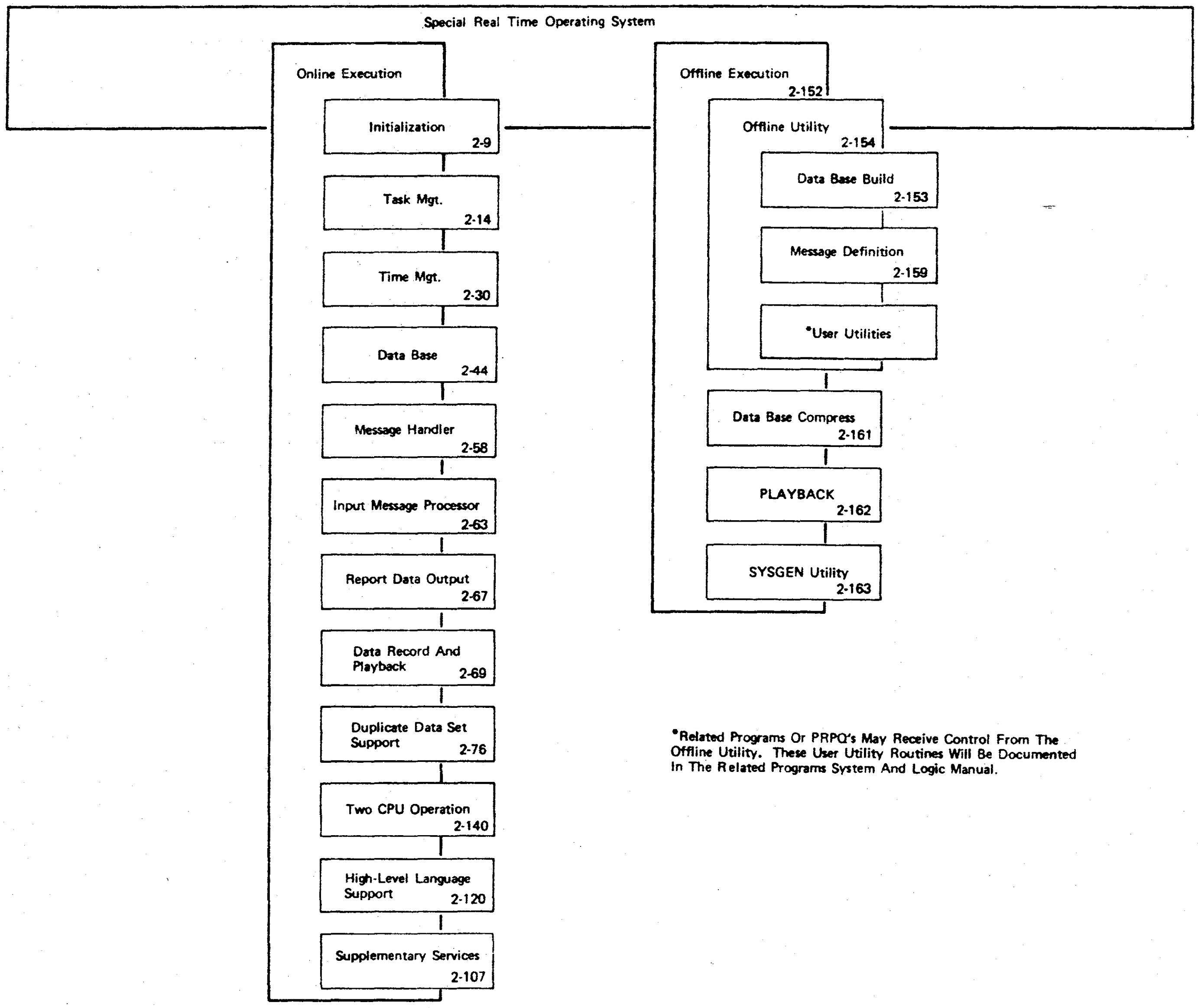
Diagram Number
Diagram Number
Message Number
PDL Segment Name Corresponds To Segment Name In Listing
Expanded Description Associated With Process Step 1 . May Include Detail Information About Input Or Output As Well As Process.
User Abend Code

2-04

Figure 2-04 (2 Of 2)

Step	Extended Description	Message And ABEND Codes	PDS Segment
1	Notes For Step 1	MSG001	SAMPLE1
2	Notes For Step 2		SAMPLE3
3	Notes For Step 3	ABEND 004	SAMPLE4

Figure 2-3 - Sample Extended Description



*Related Programs Or PRPQ's May Receive Control From The Offline Utility. These User Utility Routines Will Be Documented In The Related Programs System And Logic Manual.

Figure 2-4 - Special Real Time Operating System Overview

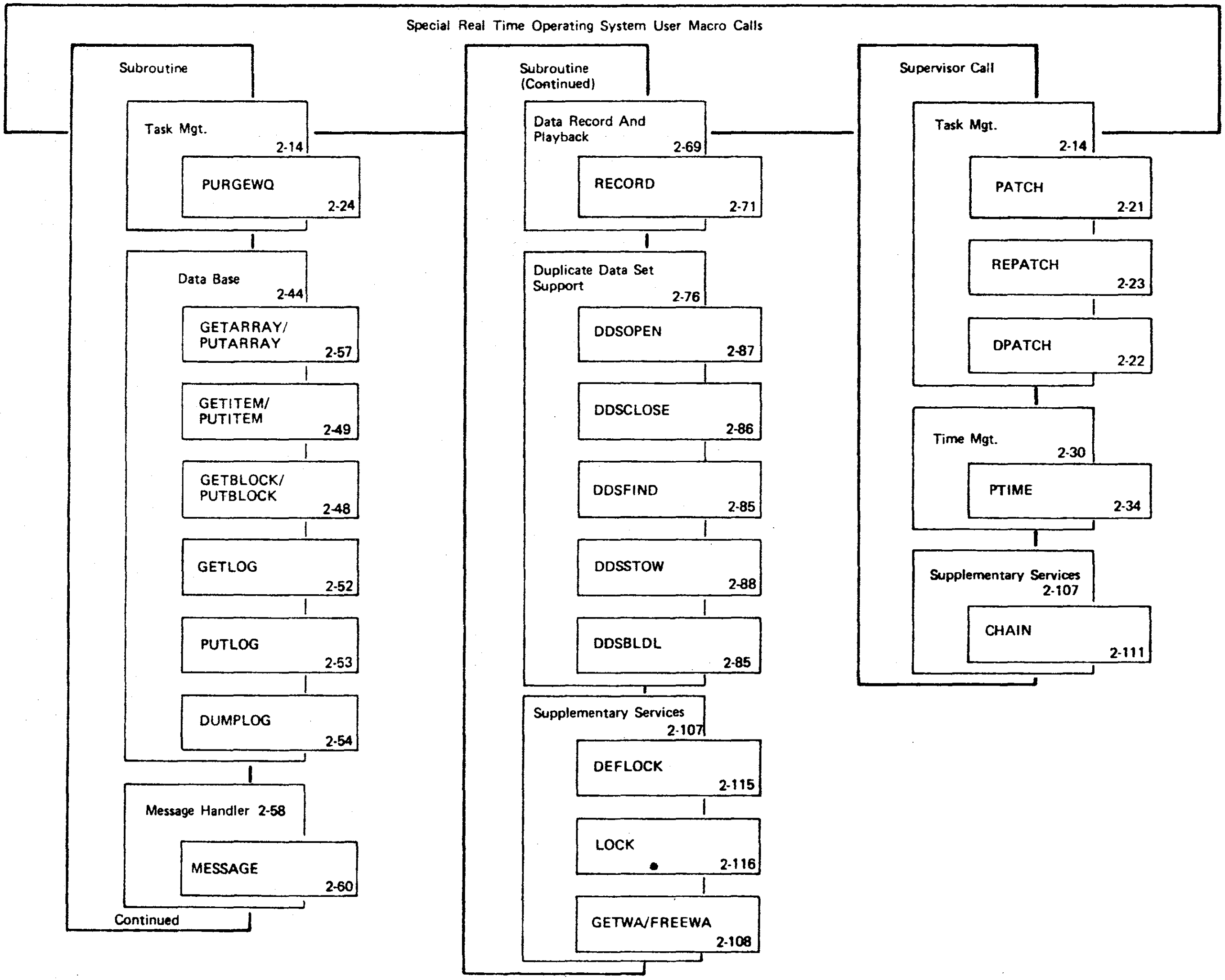


Figure 2-5 - Special Real Time Operating System Macro Calls

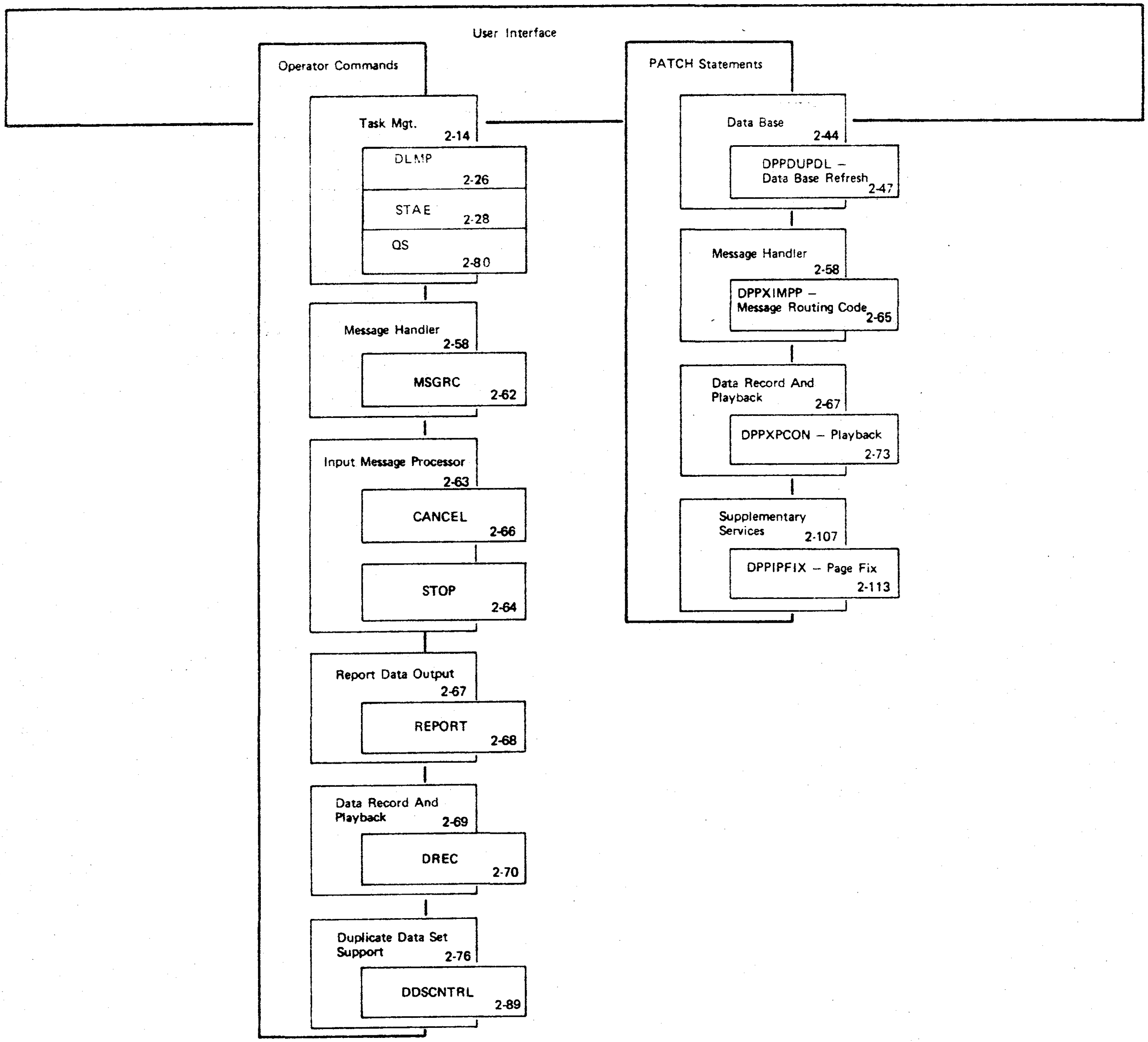
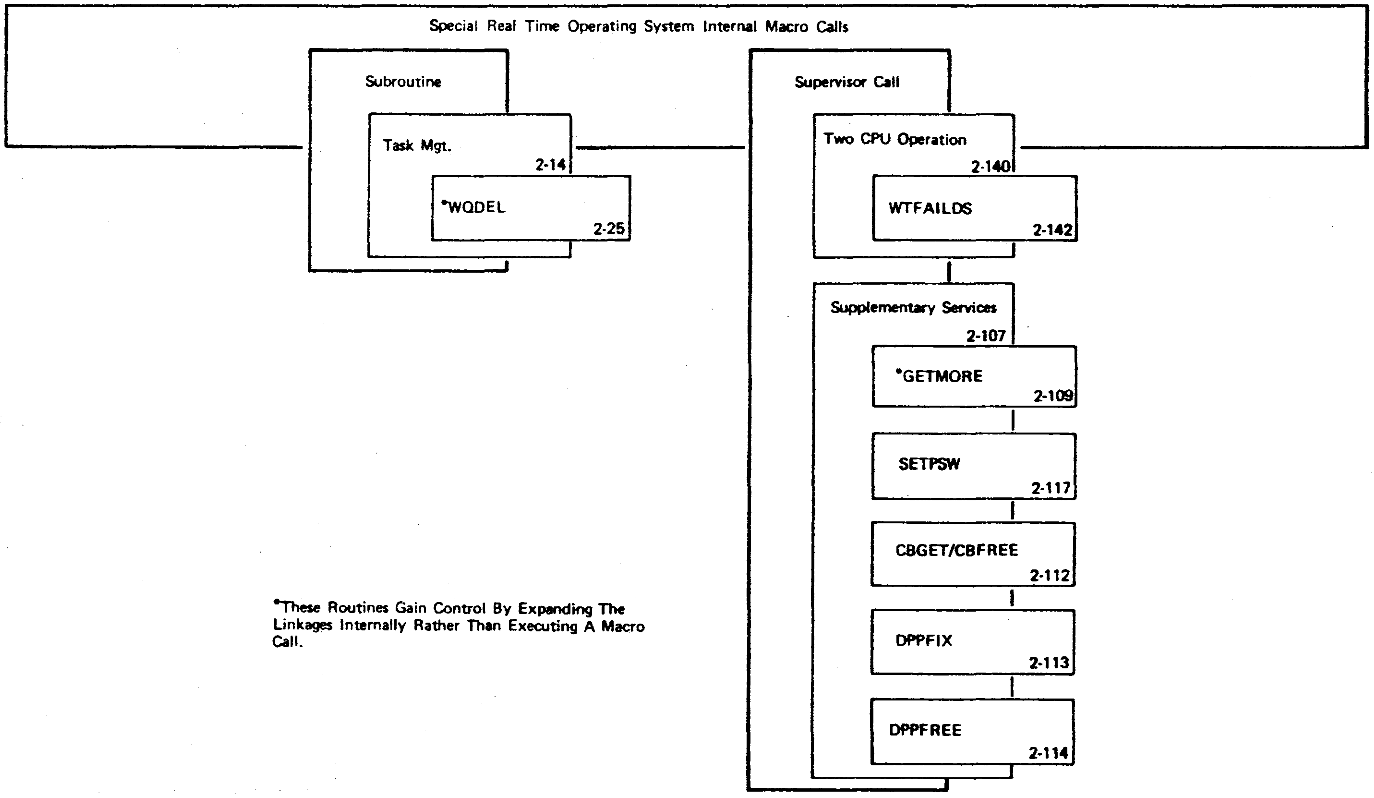


Figure 2-6 - User Interface



*These Routines Gain Control By Expanding The Linkages Internally Rather Than Executing A Macro Call.

Figure 2-7 - Special Real Time Operating System Internal Macro Calls

Initialization

The Special Real Time Operating System's initialization module, DPPINIT, is assembled during the SYSGEN procedure and contains the SYSGENed values as data constants. This module receives control from the OS/VS1 initiator whenever an EXEC statement specifying PGM=DPPINIT is executed.

The module DPPINIT references the SYSGENed values (data constants) and the SYSINIT input stream and initializes the realtime job step accordingly. Once the basic initialization has been completed by DPPINIT, control is transferred (XCTL) to the Special Real Time Operating System's system monitor routine, DPPTSMON, and the realtime job step is ready for processing to begin. Figure 2-8 provides an overview of the modules executed during the initialization process.

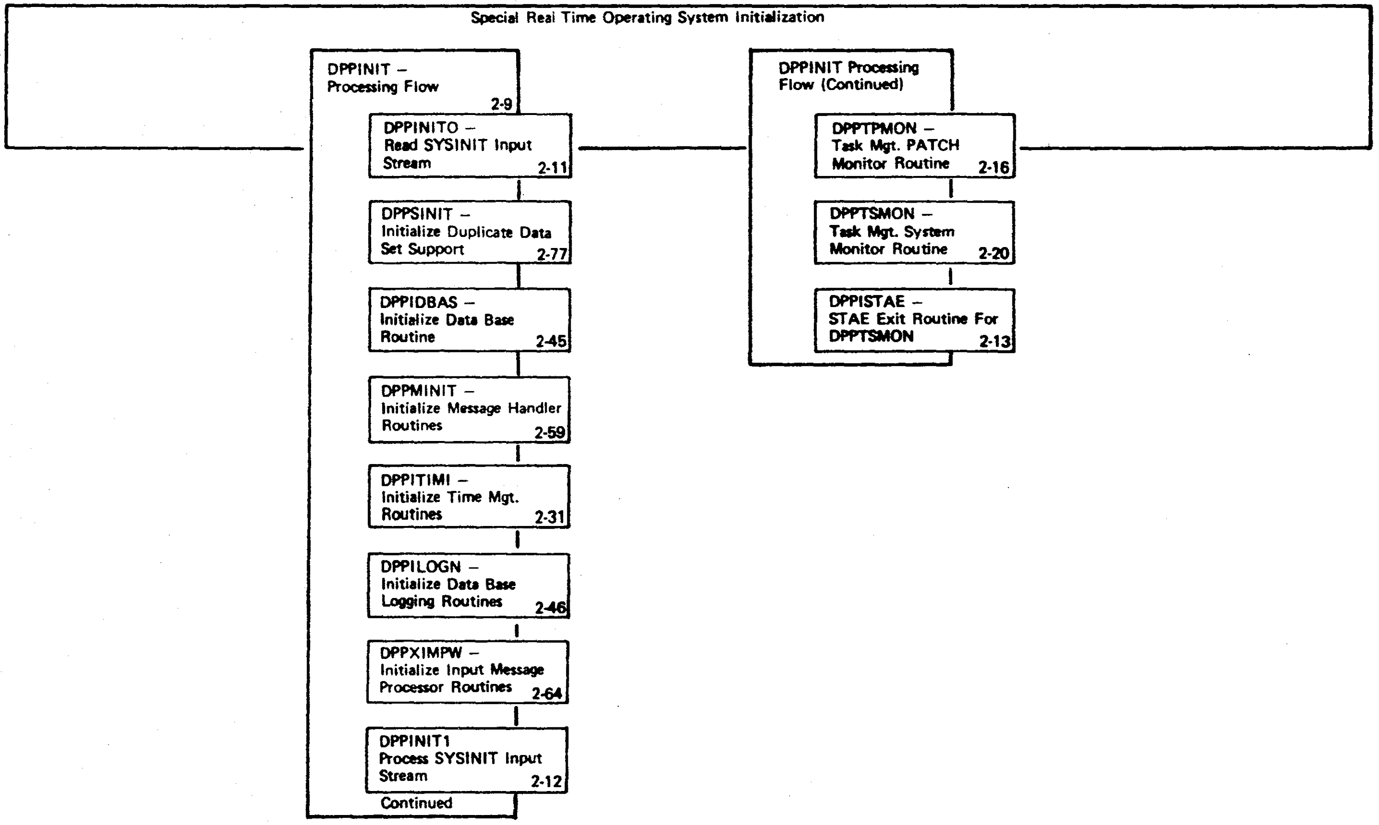

Special Real Time Operating System Initialization
DPPINIT – Processing Flow 2-9
DPPINITO – Read SYSINIT Input Stream 2-11
DPPSINIT – Initialize Duplicate Data Set Support 2-77
DPPIDBAS – Initialize Data Base Routine 2-45
DPPMINIT – Initialize Message Handler Routines 2-59
DPPITIMI – Initialize Time Mgt. Routines 2-31
DPPILOGN – Initialize Data Base Logging Routines 2-46
DPPXIMPW – Initialize Input Message Processor Routines 2-64
DPPINIT1 Process SYSINIT Input Stream 2-12
Continued
DPPINIT Processing Flow (Continued)
DPPTPMON – Task Mgt. PATCH Monitor Routine 2-16
DPPTSMON – Task Mgt. System Monitor Routine 2-20
DPPISTAE – STAE Exit Routine For DPPTSMON 2-13

Figure 2-8 - Special Real Time Operating System Initialization Overview

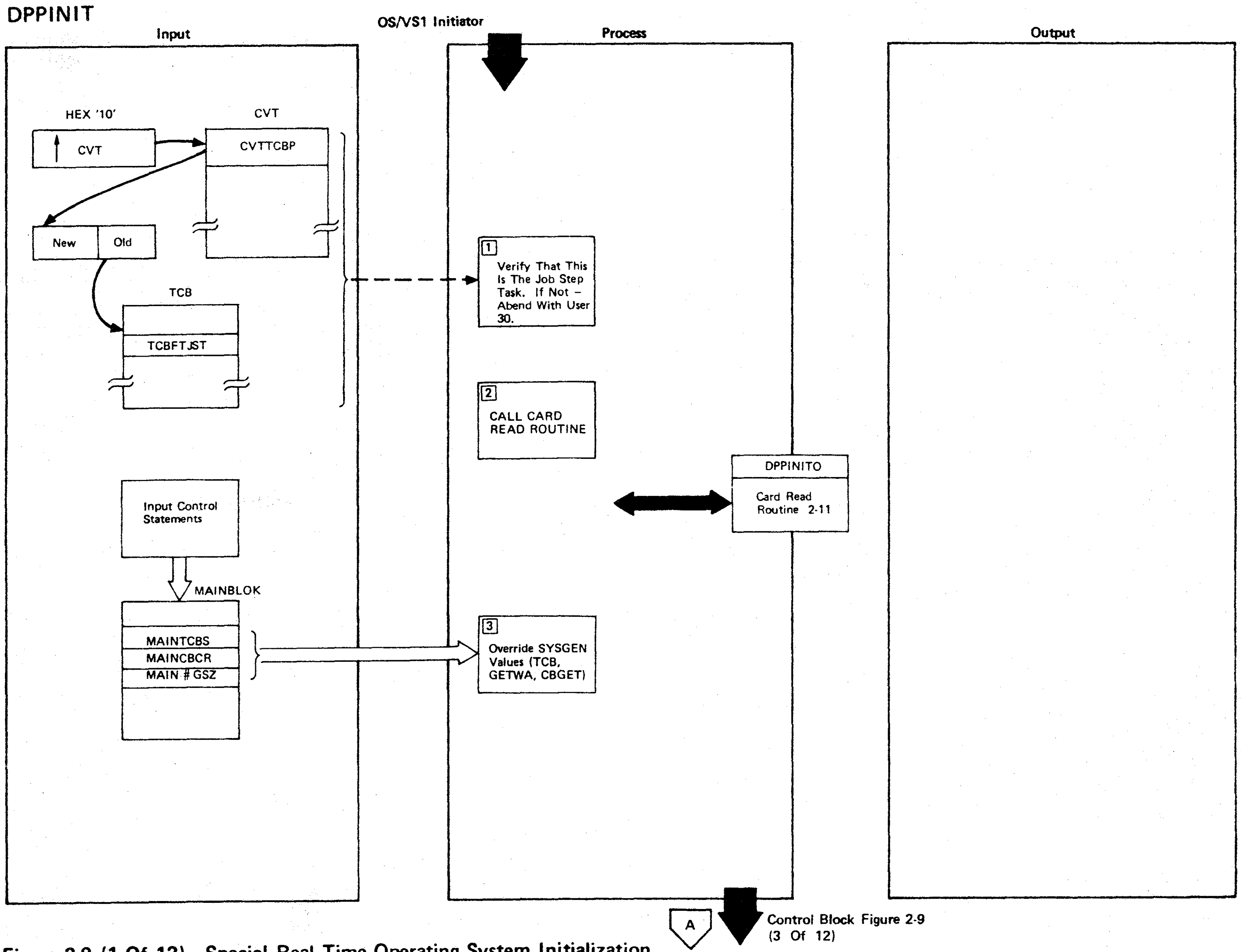


Figure 2-9 (1 Of 12) - Special Real Time Operating System Initialization

Figure 2-9 (2 Of 12)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Special Real Time Operating System initialization must run under the job step TCB. It cannot be an attached task. The CVT - new/old pointers are used to get the TCB address under which initialization is running. The TCB address is compared to the TCBFTJST address to find if it is the job step task. If it is not, the job is ABENDed with a code 30.	USER 30	DPPINIT
2	Program DPPINITO is branched to in order to have the input stream read. See Figure 2-11 for detail.		DPPINIT
3	Override SYSGEN values for number of advance TCBs and GETWA. Override CBGET value.		DPPINIT

DPPINIT

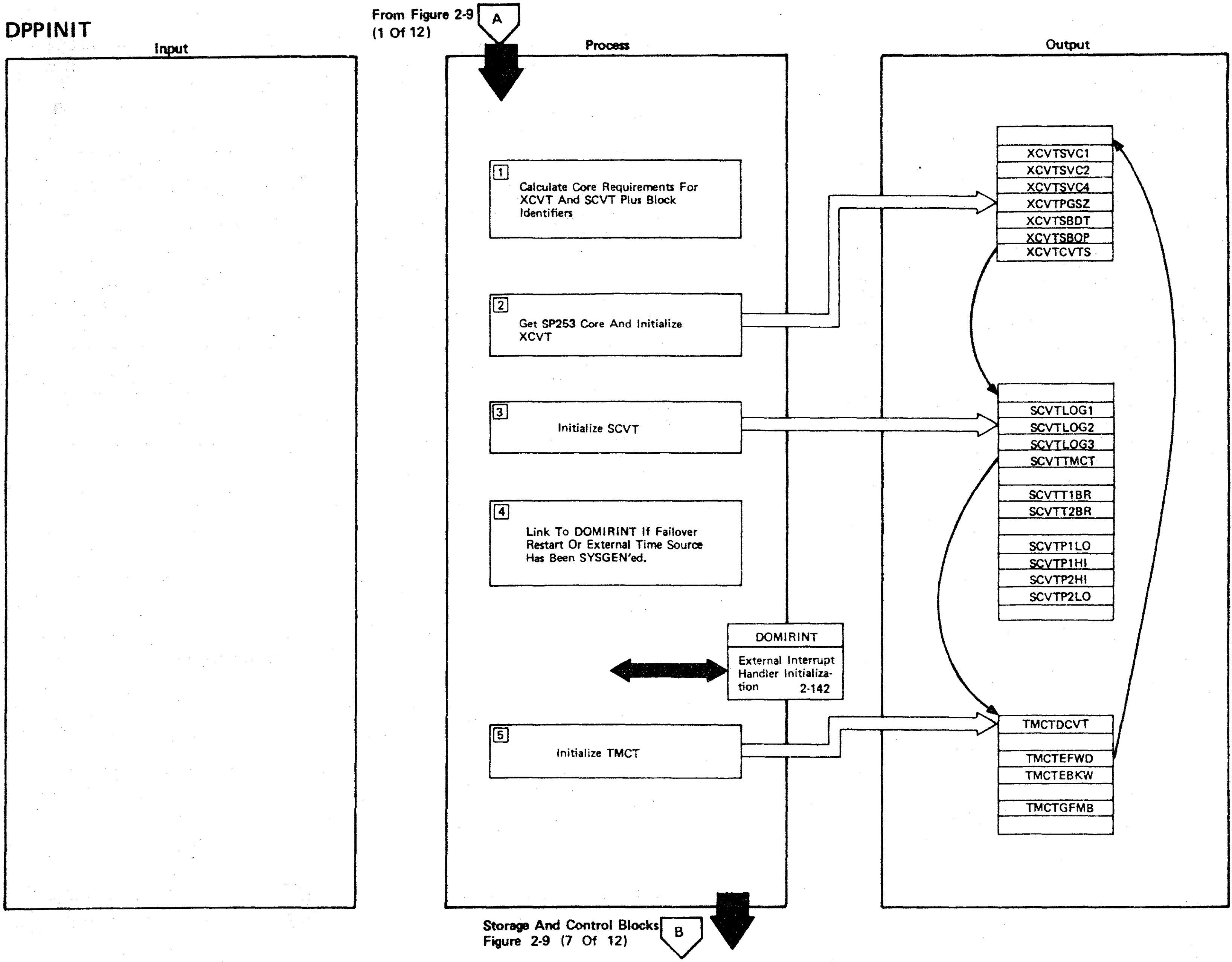


Figure 2-9 (3 Of 12) - Task Management Control Block Initialization

Figure 2-9 (4 Of 12)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The core required for the XCVT and SCVT is calculated by (XCVTLNTH + IDLNTH + SCVTLNTH + IDLNTH). The ID is an 8-byte control block identifier which precedes the control block in core and makes it easy to locate the control block in a core dump.		DPPINIT
2	The core for both the XCVT and the SCVT is obtained by one GETMAIN from subpool 253. The XCVT identifier is put ahead of the control block and then XCVT fields are initialized in the following order: XCVTSVC1 - An executable type 1 SVC instruction (OAXX) XX = SVC number XCVTSVC2 - An executable type 2 SVC instruction (OAXX) XX = SVC number XCVTSVC4 - An executable type 4 SVC instruction (OAXX) XX = SVC number XCVTSBOP - Initial flags set (XCVTPRS, XCVT1PL, XCVTCPU) XCVTPGSZ - Size of page 2K - VS1 XCVTCVTS - Pointer to SCVT		DPPINIT
3	The SCVT ID is placed ahead of the control block. The SVCT fields are then initialized in the following order: SCVTLOG1 } SCVTLOG2 } If logging is SYSGENed SCVTLOG3 } SCVTT1BR - Type 1 SVC branch table address SCVTT2BR - Type 2 SVC branch table address SCVTP1HI - Partition high address SCVTP1LO - Partition low address SCVTTMCT - Pointer to Task Management Control Table		DPPINIT
4	If either failover/restart or external time source has been SYSGENed, a link to routine DOMIRINT will be generated.		DPPINIT

Figure 2-9 (5 Of 12)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
5	A GETMAIN for the TMCT and GFMB. The TMCT ID is put ahead of the TMCT. The TMCT fields are then initialized in the following order: TMCTEFWD TMCTEBKW GETWA Type = PC dummy GFBE TMCTGFMB - Pointer to the first GFMB TMCTXCVT - Pointer to the XCVT		DPPINIT1

Intentionally Blank

Figure 2-9 (6 of 12)

DPPINIT

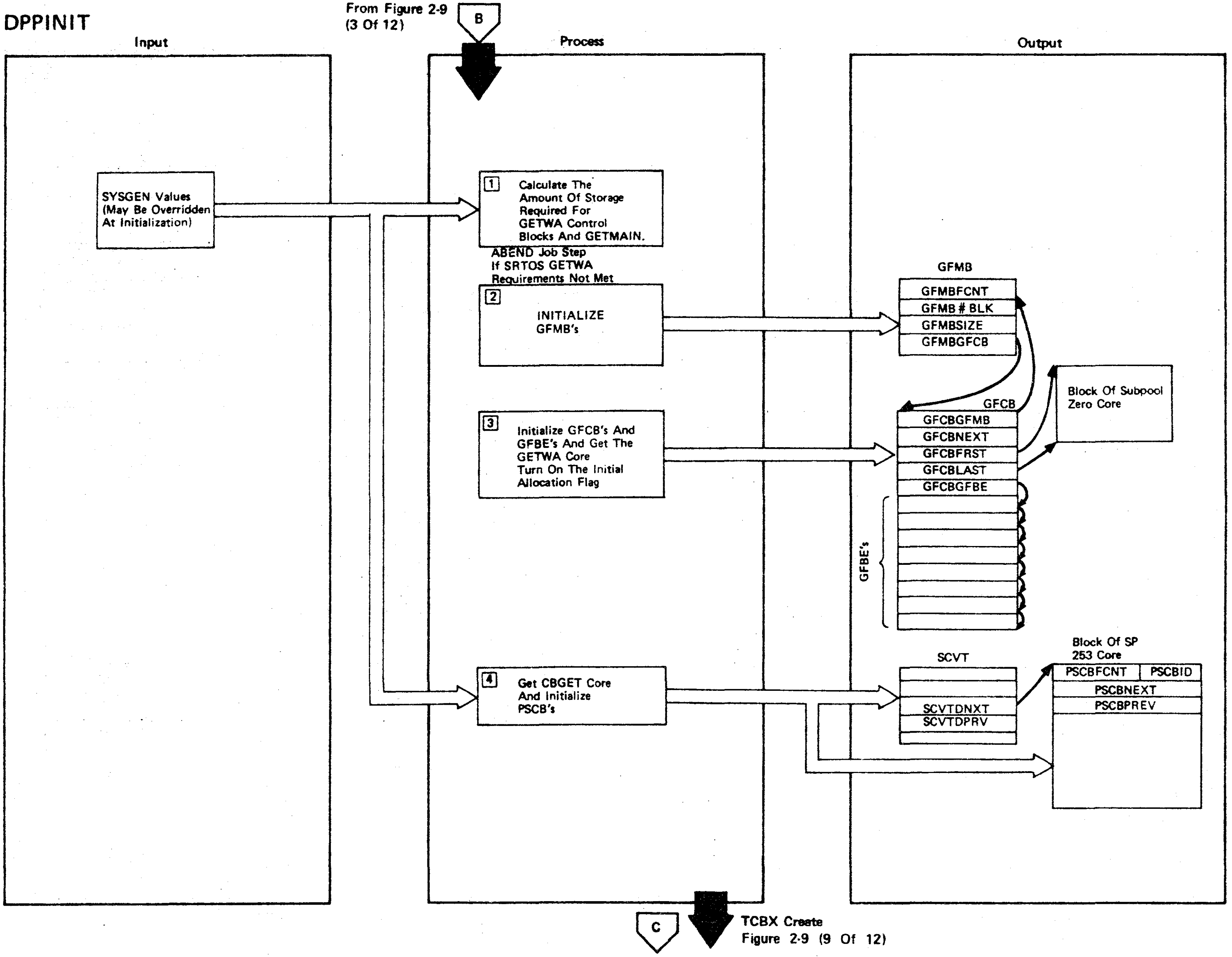


Figure 2-9 (7 Of 12) - Initialize GETWA And CBGET Storage And Control Blocks

Figure 2-9 (8 Of 12)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The amount of protected storage required for GETWA control block storage is calculated (number of sizes x GFCBLNTH) + (total number of blocks x GFBELNTH). The storage required is GETMAINed from subpool 253. If a GETWA size of at least 1024 bytes is not requested, the job step is ABENDed with a code 46.	USER 46	DPINIT1
2	The following fields are initialized in the GFMB GFMBSIZE - Size of GETWA blocks GFMBFCNT - Free count of GETWA blocks GFMB#BLK - Initial number of blocks requested GFMBGFCB - Pointer to corresponding GFCB GFMBID - ID (0, 1, 2, 3...31) maximum 31		DPINIT1
3	The GFCB is initialized in the following manner GFCBGFMB - Pointer to corresponding GFMB GFMBGFBE - Pointer to first GFBE (free queue) - All GFBEs are then queued to this free queue. GFCBFRST - Low address of associated SP zero core GFCBLAST - High address of associated SP zero core The initial allocation flag (GFCBINIT) is turned on.		DPINIT1
4	The amount of CBGET core is calculated if no value is given at initialization time (CBGET statement) and core is obtained from subpool 253. A Protected Storage Control Block (PSCB) is built in the first 12 bytes of obtained core and is backward (PSCBREV) and forward (PSCBNEXT) chained to the dummy PSCB in the SCVT. The number of 32 byte blocks available is calculated from the number stored in the PSCBFCNT field.		DPINIT2

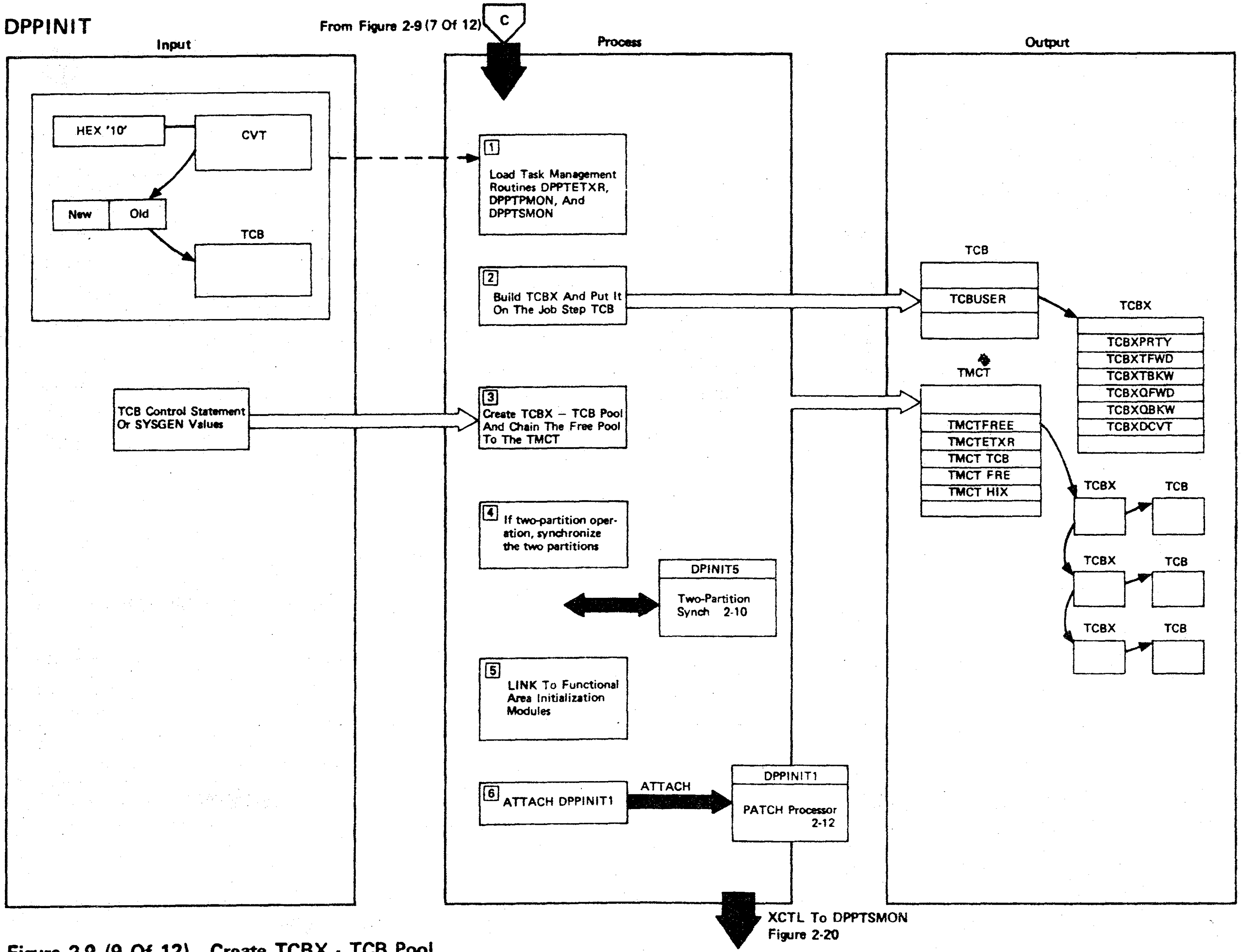

DPPINIT
From Figure 2-9 (7 Of 12)
C
Input
Process
Output
HEX '10'
CVT
New
Old
TCB
TCB Control Statement Or SYSGEN Values
1 Load Task Management Routines DPPTETXR, DPPTPMON, And DPPTSMON
2 Build TCBX And Put It On The Job Step TCB
3 Create TCBX — TCB Pool And Chain The Free Pool To The TMCT
4 If two-partition operation, synchronize the two partitions
DPINIT5
Two-Partition Synch 2-10
5 LINK To Functional Area Initialization Modules
6 ATTACH DPPINIT1
ATTACH
DPPINIT1
PATCH Processor 2-12
XCTL To DPPTSMON Figure 2-20
TCB
TCBUSER
TCBX
TCBXPRTY
TCBXTFWD
TCBXTBKW
TCBXQFWD
TCBXQBKW
TCBXDCVT
TMCT
TMCTFREE
TMCTETXR
TMCT TCB
TMCT FRE
TMCT HIX
TCBX
TCB
TCBX
TCB
TCBX
TCB

Figure 2-9 (9 Of 12) - Create TCBX - TCB Pool

Figure 2-9 (11 Of 12)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
5	Special Real Time Operating System subroutines are loaded or linked in the following order: LOAD DPPTSTAE - Store address in SCVTSTAE LOAD DPPTPWQE - Store address in SCVTPWQE LOAD DPPXDEFL - Store address in SCVTDEFL - LOCK LOAD DPPTGWFW - Store address in SCVTGWBS LOAD DPPXLOCK - Store address in SCVTLOCK - DEFLOCK LINK DPPSINIT - If DDS SYSGENed LOAD DPPSOP1 - If DDS SYSGENed LOAD DPPSCL1 - If DDS SYSGENed LOAD DPPSBF1 - If DDS SYSGENed LOAD DPPSST1 - If DDS SYSGENed LOAD DPPXDRCX - Store address in SCVTREC - Data recording LINK DPPIDBAS - Link to data base initialization LINK DPPMINIT - Link to message handler initialization LINK DPPITIMI - Link to time management initialization LINK DPPILOGN - Link to logging initialization if SYSGENed ATTACH DPPXIMPW - Input message processor LOAD DPPIPFRE - Store address XCVTPFRE LOAD DPPISTAE - Job step STAE routine		DPINIT3
6	Initialization is complete so the PATCH stream processor (DPPINIT1) is attached, then control is passed (XCTL) to the system monitor (DPPTSMON). ATTACH DPPINT1 XCTL DPPTSMON.		DPPINIT

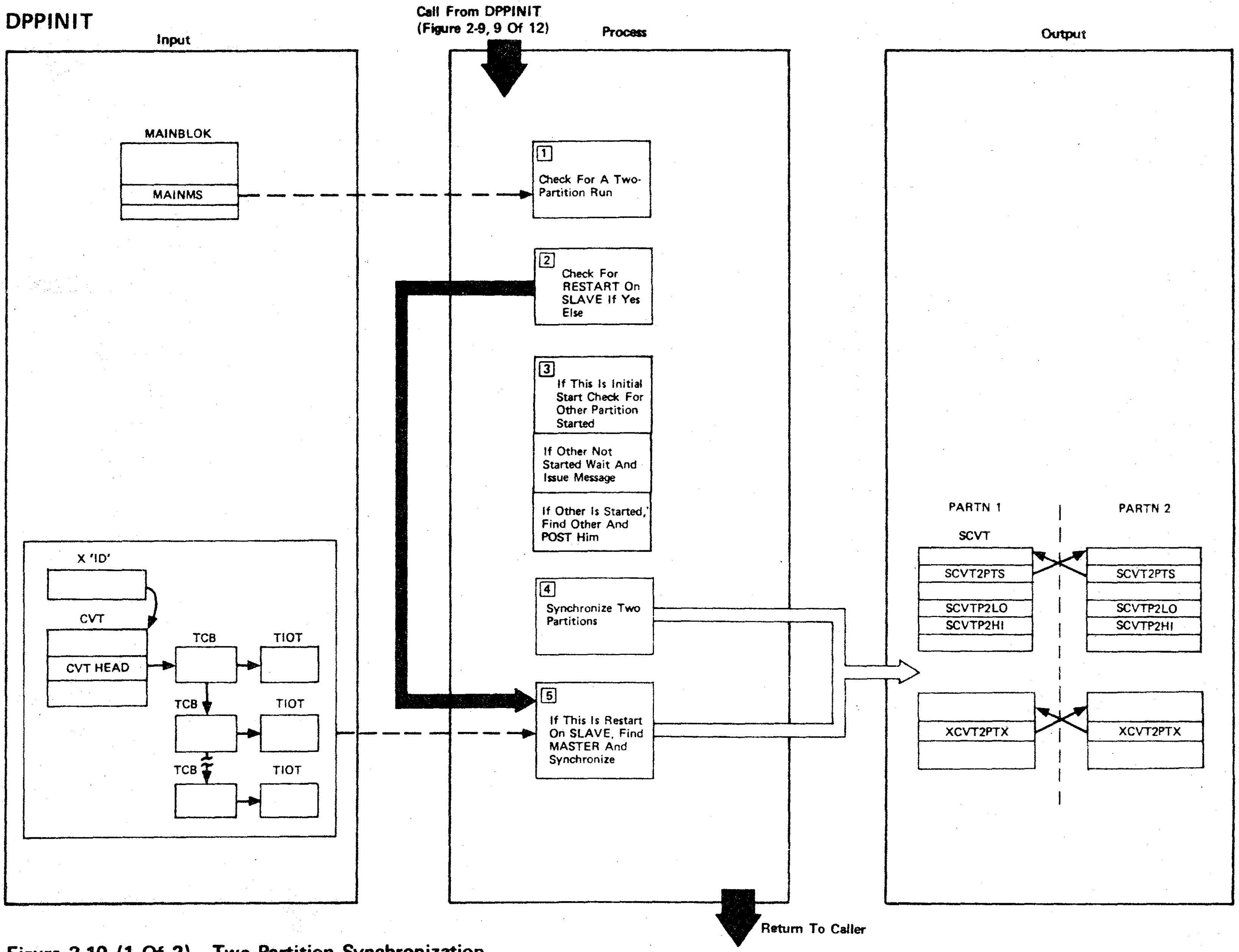
Figure 2-9 (10 Of 12)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The Special Real Time Operating System task management routines are brought into virtual storage via the LOAD macro. LOAD DPPTSMON LOAD DPPTETXR, store ETXR address in TMCTETXR LOAD DPPTPMON, store interface entry point in SCVTPMON.		DPINIT3
2	A TCBX is created and initialized and chained to the job step task TCB. The TCBX fields are initialized as follows: TCBXPRTY - TCB's dispatching priority TCBXTFWD - { Dummy GETWA type at GFBE TCBXTBKW - TCBXQFWD - { Dummy GETWA type at GFBE TCBXQBKW - TCBXDCVT - Pointer to the XCVT If enough CBGET storage cannot be obtained, the job step task is terminated with a code 33.	USER 33	DPINIT3
3	A TCB pool is created by ATTACHing DPPTPMON for the number of advance TCBs. A TCBX is created for each TCB (as in step 2) and the free chain is chained to the TMCTFREE chain.		DPINIT3
4	The two partition flags in the MAINBLOK are tested (MAINMSTR, MAINSLAV); if either is on, the two partitions are synchronized (see Figure 2-10).		DPINIT5

Intentionally Blank

Figure 2-9 (12 of 12)

DPPINIT



1 Check For A Two-Partition Run

2 Check For RESTART On SLAVE If Yes Else

3 If This Is Initial Start Check For Other Partition Started

If Other Not Started Wait And Issue Message

If Other Is Started,' Find Other And POST Him

4 Synchronize Two Partitions

5 If This Is Restart On SLAVE, Find MASTER And Synchronize

Figure 2-10 (1 Of 2) - Two Partition Synchronization

Figure 2-10 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The MAINBLOCK MAINMS flags are checked for the presence of a MASTER or SLAVE statement in the input stream. If none exists, the synchronization routine is bypassed.		DPINIT5
2	An ENQ for MASTER jobname - MASTER jobname is issued, if the resource is available, this is an initial start. If it is not available, this is a restart of a SLAVE partition, processing continues at step 5.		DPINIT5
3	An ENQ is issued on MASTER jobname - SLAVE jobname to find if the other partition is started. If the resource is available the other partition has not started, a message is issued and the partition WAITs. If the resource is not available, the other partition has started and it is WAITing. The TCB ready queue is searched, looking for the jobname in the TCB's TIOT; when the other job is found, its XCVT is posted with this partition's XCVT address.	DPP046I	DPINIT5
4	Each partition gets the other's low and high partition addresses and puts these in his own SCVT, and gets the other's SCVT address and puts it in his own SCVT.		DPINIT5
5	If this is a restart on a SLAVE, the MASTER is located via the TCB ready queue and if he is not currently ABENDing and does not already have a SLAVE partition, he is given the SLAVE's low and high partition boundaries, and the SLAVE's SCVT and XCVT address. The SLAVE gets the MASTER's low and high partition boundaries, and the MASTER's SCVT and XCVT addresses, the two partition bit is set on in each XCVT, and the resync bit is set on in the MASTER's XCVT and initialization continues.	USER 36 USER 42 USER 43 USER 44	DPINIT5

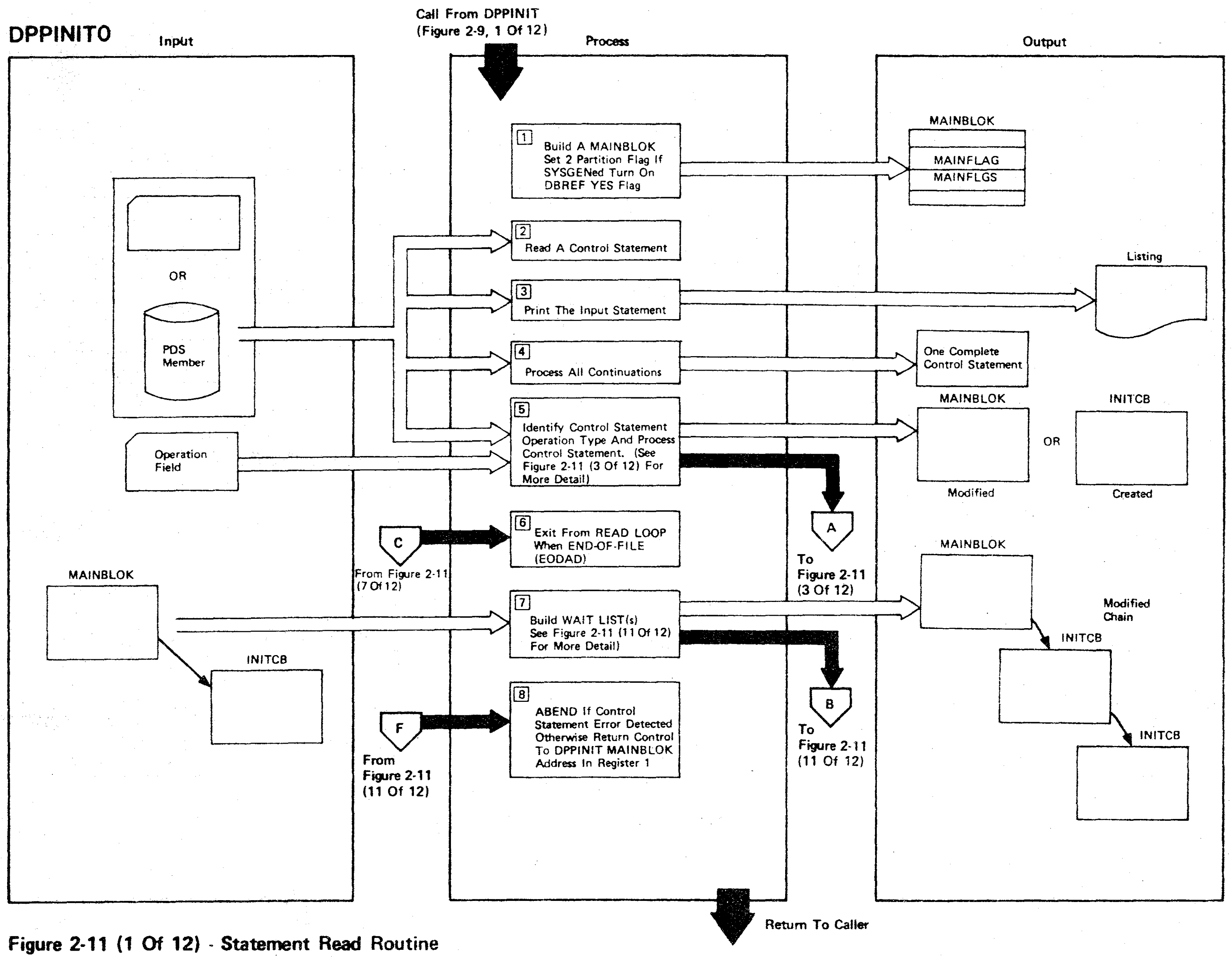


Figure 2-11 (1 Of 12) - Statement Read Routine

Figure 2-11 (2 of 12).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	A MAINBLOK is built in subpool 0.		DPPINITO
2	A read loop is established to read control statements. The only exit from the read loop is end-of-file (EODAD) on SYSINIT. The label is moved to the work area. If column 1 is nonblank and if the data in column 1 is an asterisk (*), the statement is a comment statement, it is written to the INITLIST data set and the next card is read. If it is not a comment statement, the operation is moved to the work area, then the operands are moved to the work area. All comments and blanks used as delimeters are removed, and only meaningful information is moved to the work area. Blanks within the PARAM field are kept. A flag is set to indicate continuation if column 72 is nonblank or the last data column contained a comma. If column 72 was nonblank and the last data column was not a comma, a flag is set to indicate that no more operands are expected. If an error is found in control statement, issue an error message.	DPP836I DPP800I DPP801I DPP802I DPP822I USER 40 DPP045I	DPPINITO DPINIT06
3	The input control statement is written to the SYSLIST data set.		DPPINITO
4	Continuation cards are read until there are no more continuations expected. If the maximum number of operands is not exceeded, the operands are moved to the work area.	DPP804I DPP822I	DPINIT03
5	The control statement operation type is identified. (See Figure 2-11 (3 of 12) for detail.)		
6	At end-of-file, control is passed to program label BLDWTLST.		DPINITO
7	See Figure 2-11 (11 of 12) for description of Build Wait List routine (BLDWTLST).		DPINIT05
8	If any errors were detected during control statement processing, the job step is ABENDed with a code 34; otherwise, control is returned to DPPINIT.	USER 34	DPINIT05

DPPINITO

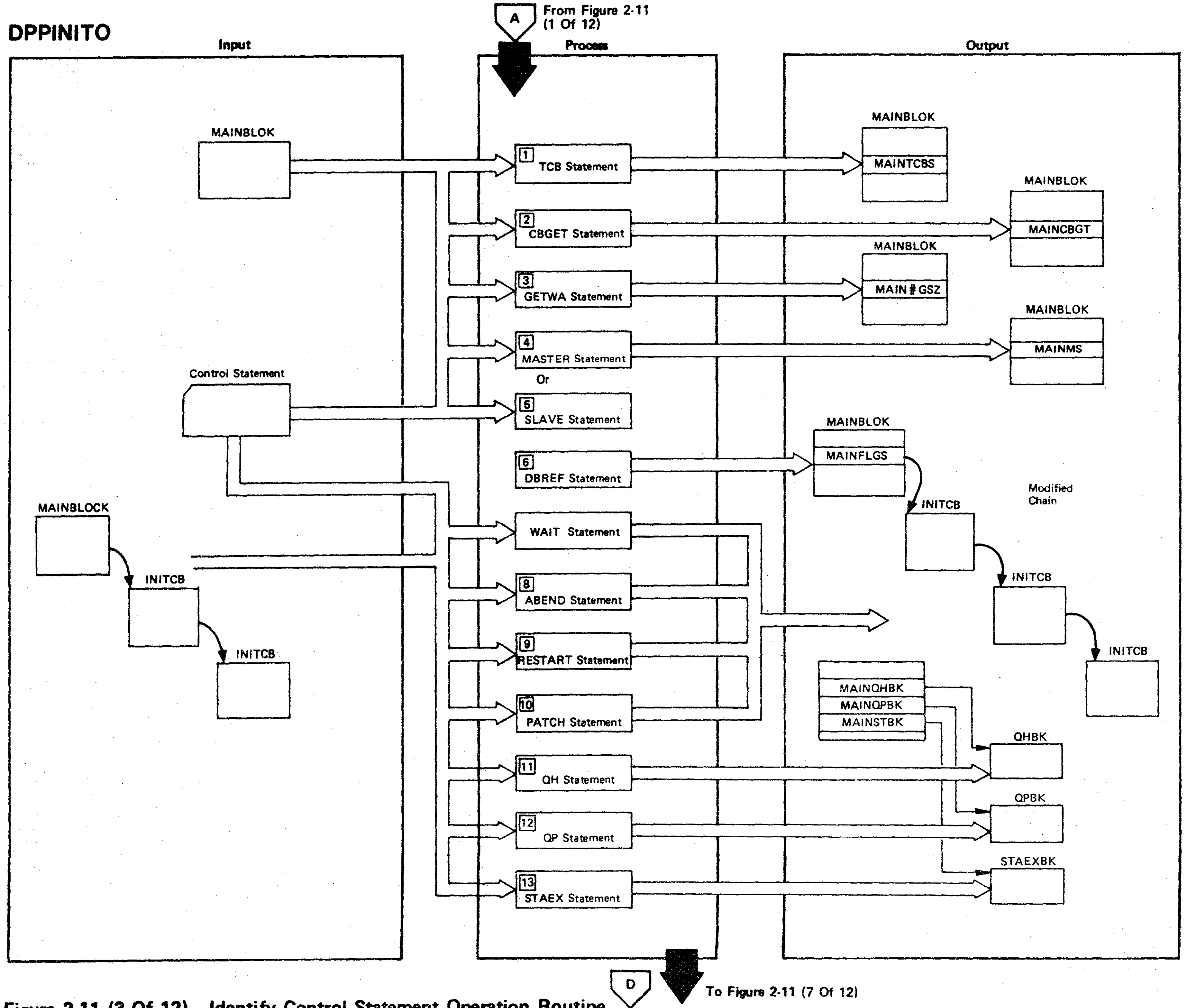


Figure 2-11 (3 Of 12) - Identify Control Statement Operation Routine

Figure 2-11 (4 of 12).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Ensure TCB field contains all decimal data. Convert data and store value in MAINTCBS.	DPP834I DPP805I	DPINIT04
2	Ensure CBGET field contains all decimal data. Convert data and storage value in MAINCBGT.	DPP829I DPP805I	DPINIT04
3	Ensure all decimal GETWA input and number suboperands do not exceed 32. Move the converted data to the MAINBLOK and sort the entries by GETWA size. Ensure that the number of blocks is not greater than 4095 and the size is not greater than 30760. Ensure blocks which are greater than 2K are 2K multiples.	DPP037I DPP038I DPP039I DPP040I DPP041I DPP042I DPP043I	
4/5	If two partition SYSGENed, accept MASTER or SLAVE statements. If MAINMSTR and MAINSLAV are both off, no previous MASTER or SLAVE statement has been encountered in the input stream. The MASTER= or SLAVE= operand is verified and if it is valid, the job name is moved to the MAINNAME field and the appropriate flag (MAINMASTR if MASTER or MAINSLAV if SLAVE) is turned on.	DPP806I DPP813I DPP830I DPP845I	DPINIT04
6	If DBREF NO request, turn off the refresh flag (MAINRIMT) in the MAINBLOK. If YES, the flag is left alone as it is already set.	DPP802I DPP805I	DPINIT04
7	Build an INITCB, chain it on the chain, and turn on the INITWAIT flag to identify this is a WAIT control block, locate the INITCB with the given label and verify that it is a PATCH block. If it is, calculate the address of the PATCH blocks INITECB field and put it in the WAIT blocks ECB field.	DPP807I	DPINIT04
8	Build an INITCB, chain it on and turn on the INITABND flag to identify this as an ABEND control block. If DUMP was requested, turn on the INITDUMP flag. If a time was specified, convert it and put it in the INITECB field, otherwise put the default time (30 seconds) in the INITECB field.	DPP833I DPP832I DPP831I	DPINIT04

Figure 2-11 (5 of 12).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
9	Build and chain an INITCB and turn on the INITWRST flag to identify this as a RESTART control block if no previous RESTART WRITE statement has been read. Set WRITE, PROBE, CMON, and CANCEL flags as required.	DPP805I DPP808I	DPINIT04
10	See following pages for detailed description of PATCH statement processing.		
11	Build QPBK from data on QP statement and chair. Add to main QPBK chair.	DPQ805I DPP813I DPP814I DPP828I DPP848I DPP849I DPP852I DPP853I DPP854I DPP857I	DPINITOA
12	Build QHBK from data on OH statement and add to main QHBK chair.	DPP801I DPP811I DPP813I DPP848I DPP852I DPP856I	DPINITOA
13	Build STAX BX from data on STAEX statement and add to main STBK chair.	DPP801I DPP813I DPP849I DPP853I DPP854I DPP855I	

Intentionally Blank

Figure 2-11 (6 of 12)

DPPINITO

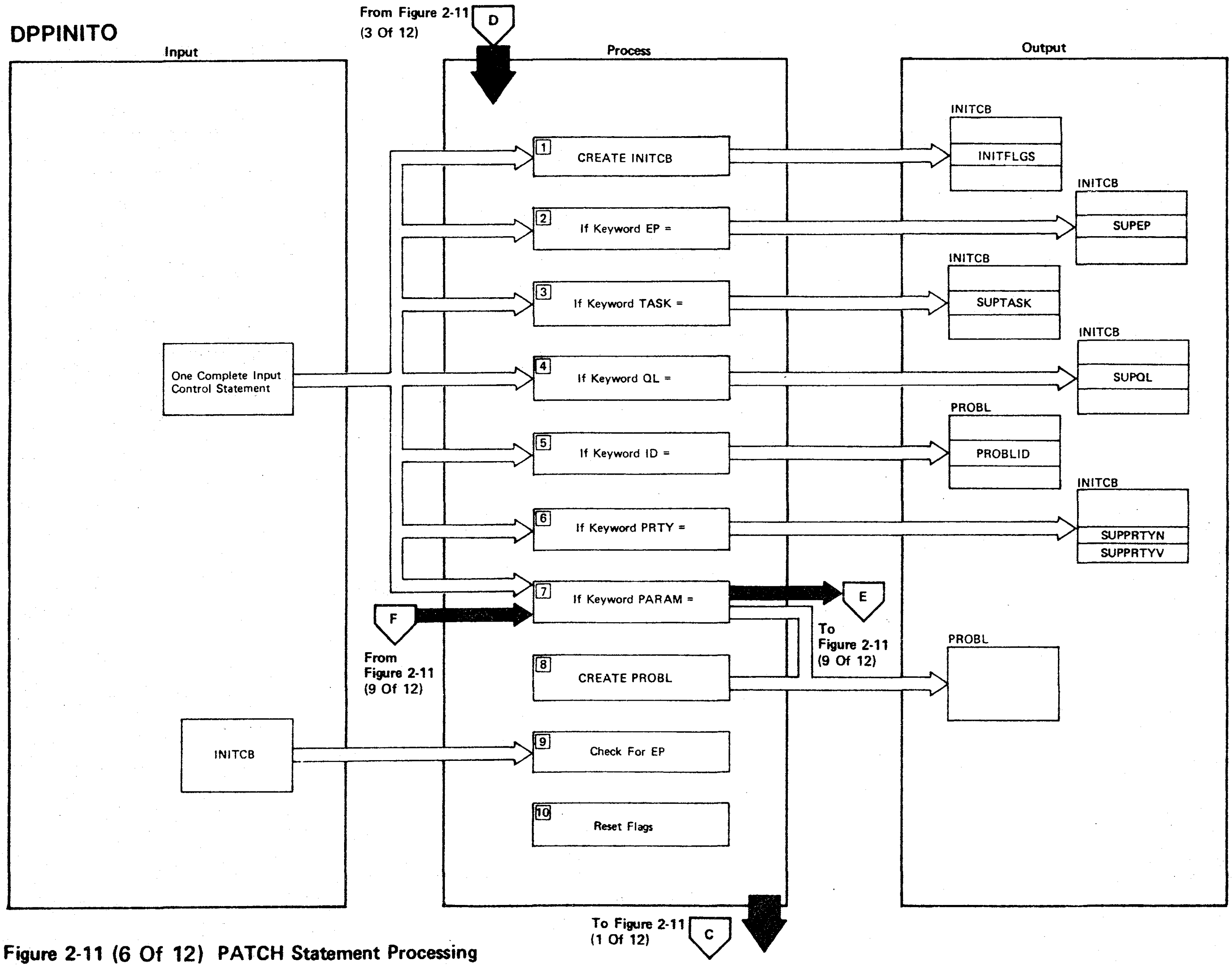


Figure 2-11 (6 Of 12) PATCH Statement Processing

Figure 2-11 (8 of 12).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Create and chain an INITCB. Turn on the INITPTCH flag to identify this as a PATCH control block and move control statement label to INITLABL.		DPINIT02
2	If no EP=keyword previously processed for this PATCH statement, turn on the PTCHEP flag and move the EP name to the SUPL. The SUPL is part of the INITCB.	DPP809I DPP817I	DPINIT02
3	If no TASK=keyword previously processed for this statement, turn on the PTCHTASK flag and move the TASK name to the SUPL.	DPP810I DPP817I	DPINIT02
4	If no QL=keyword previously processed for this statement, turn on the PTCHQL flag. Validity check the QL data, convert it, and put the converted value in the SUPL.	DPP826I DPP811I DPP817I	DPINIT02
5	If no ID=keyword previously processed, turn on the PTCHID flag. Validity check the ID value, convert the value, and save it to be moved later to the PROBL.	DPP827I DPP812I DPP817I	DPINIT02
6	If no PRTY=keyboard previously processed, the PTCHPRTY flag is turned on. If the first character of the operand is a left parenthesis, the operand is of the format (job name, prty). The job name is moved to the SUPL and the priorty value is validity checked, converted, and moved to the SUPL. If the first character is not a left parenthesis, the operand is of the format JOBSTEP-n. The priority reference value is validity checked, converted, and moved to the SUPL.	DPP814I DPP815I DPP828I DPP816I DPP817I	DPINIT02
7	See Figure 2-11 (9 of 12) for processing description of PARAM.		DPINIT01
8	If no PROBL exists (no PARAM=keyword), create a PROBL. Move the ID to the PROBL.		DPINIT02
9	Check PTCHFLGS for PTCHEP flag to ensure EP=specified.	DPP835I	DPINIT02
10	Reset PTCHFLGS.		DPINIT02

DPPINITO

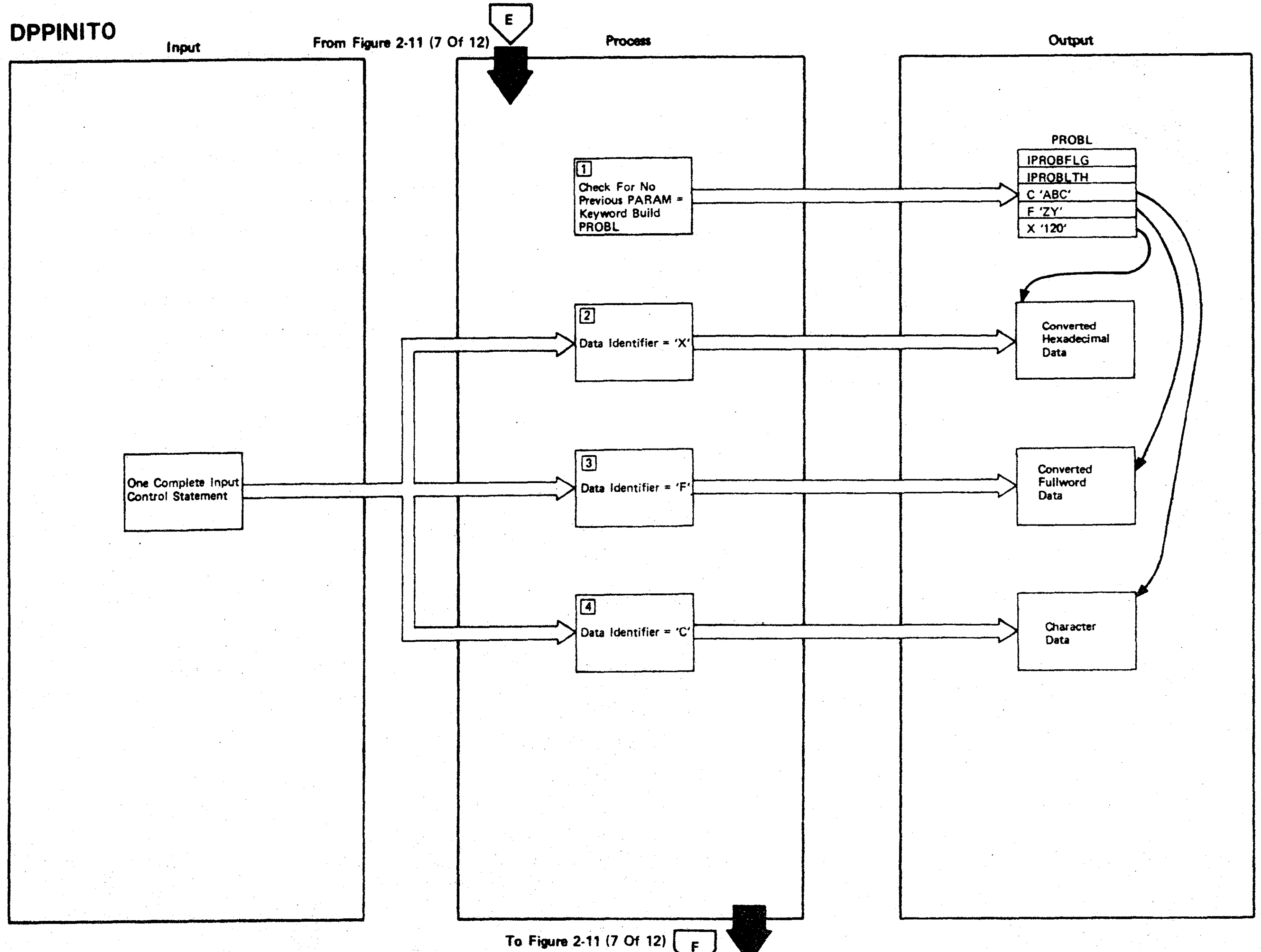


Figure 2-11 (9 Of 12) - Build Wait List Routine

Figure 2-11 (10 of 12).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If no previous PARAM=keyword has been processed, turn on the PTCHPRAM flag. The first data character is checked. If it is a left parenthesis, the operand is scanned, and the quote characters are counted. If the quotes are balanced (even number), a PROBL is created. If no right parenthesis, then issue error message.	DPP824I DPP822I DPP828I	DPINIT01
2	If data type is X, validity check the data, convert it, get storage for the data, and move the converted data to the storage area. The address of the converted data is then placed in the PROBL along with the data length.	DPP818I DPP819I	DPINIT01
3	If the data is F, four bytes of main storage is obtained, and the data is checked to see if a sign was specified. The data is converted, and if a minus sign was specified, the data is complemented. The converted data is placed in the obtained storage, and the address and length are placed in the PROBL.	DPP818I	DPINIT01
4	If the data type is C, storage is obtained and the character data is moved to the storage. The address and length of the storage are placed in the PROBL.	DPP820I DPP821I DPP818I	DPINIT01

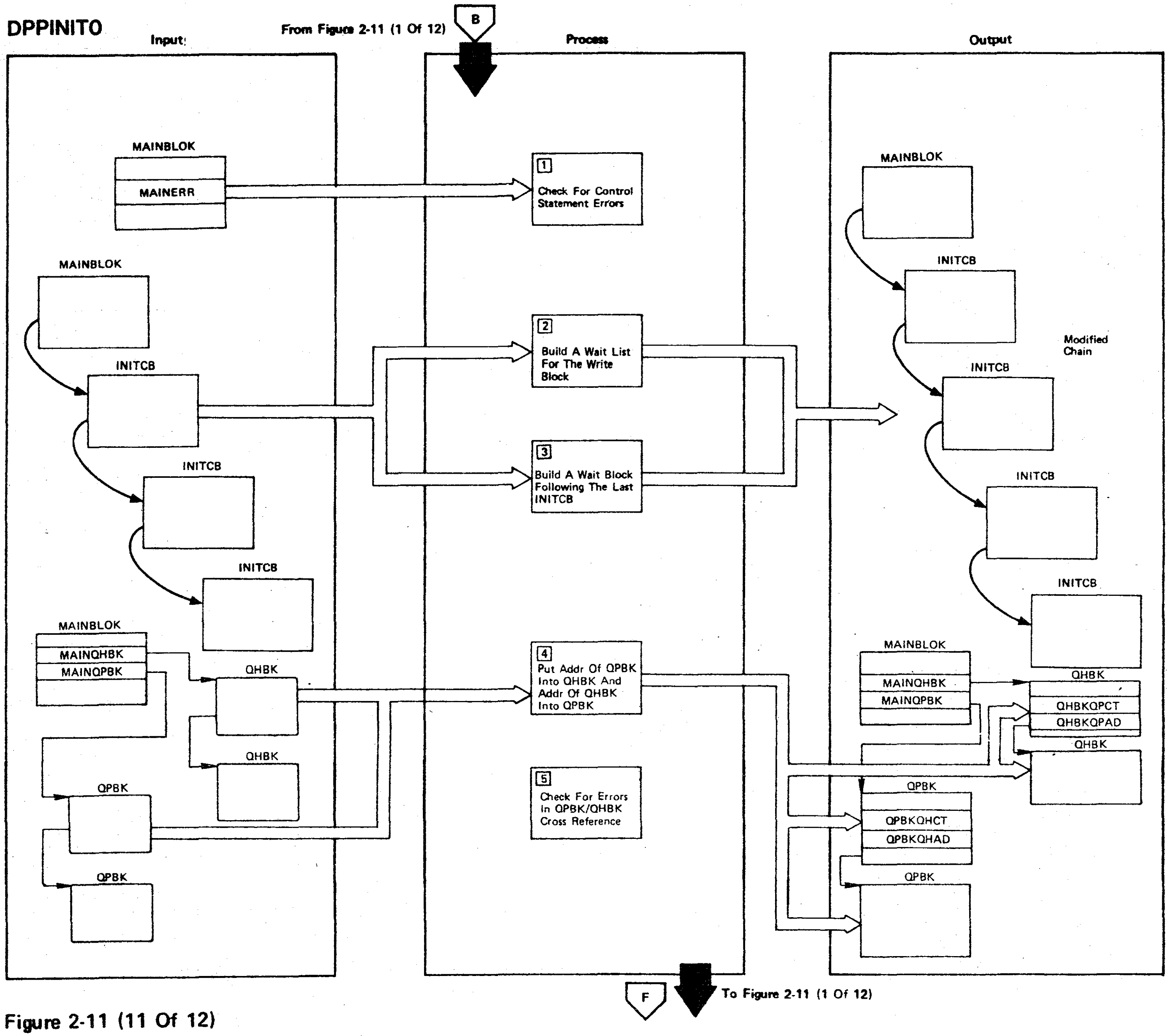


Figure 2-11 (11 Of 12)

Figure 2-11 (12 of 12).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the MAINERR flag is on, a control statement error was detected and the job step is ABENDed with code 34, otherwise, the wait list(s) are built.		DPINIT05
2	A count is made of the number of PATCH blocks. If there are no PATCH blocks, the job step is ABENDed with a code 40. If a WRITE block exists, the PATCH blocks preceding the WRITE block are counted, and a wait list is created pointing to the INITECB field of each PATCH block. The wait list address is placed in the WRITE blocks INITECB field, and the count of entries in the wait list is placed in the INITWTCT field.		DPINIT05
3	All PATCH blocks following the WRITE block (or all PATCH blocks if no WRITE blocks exist) having the PARAM= (with greater than 8 bytes for a PROBL length) parameter are counted and a wait list entry is created for each. An INITCB is created and chained to the end of the INITCB chain, the INITWAIT and INITWLST flags are turned on. The new block is pointed to the wait list by the INITECB field, and the count of the number of entries in the wait list is put in the INITWTCT field.		DPINIT05
4	The address of the QH blocks reference by each QP block is stand into the QP block and the addr. of the QP block is stored into the QH block. If more than 21 connections to any QH block, or referenced QH name not found, output message and seter for flag.	DPP846I DPP847I DPP857I	DPINITOS
5	If any errors found in plan QP/QH cross reference check, abend code 34.		

LICENSED MATERIAL — PROPERTY OF IBM

DPPINIT1

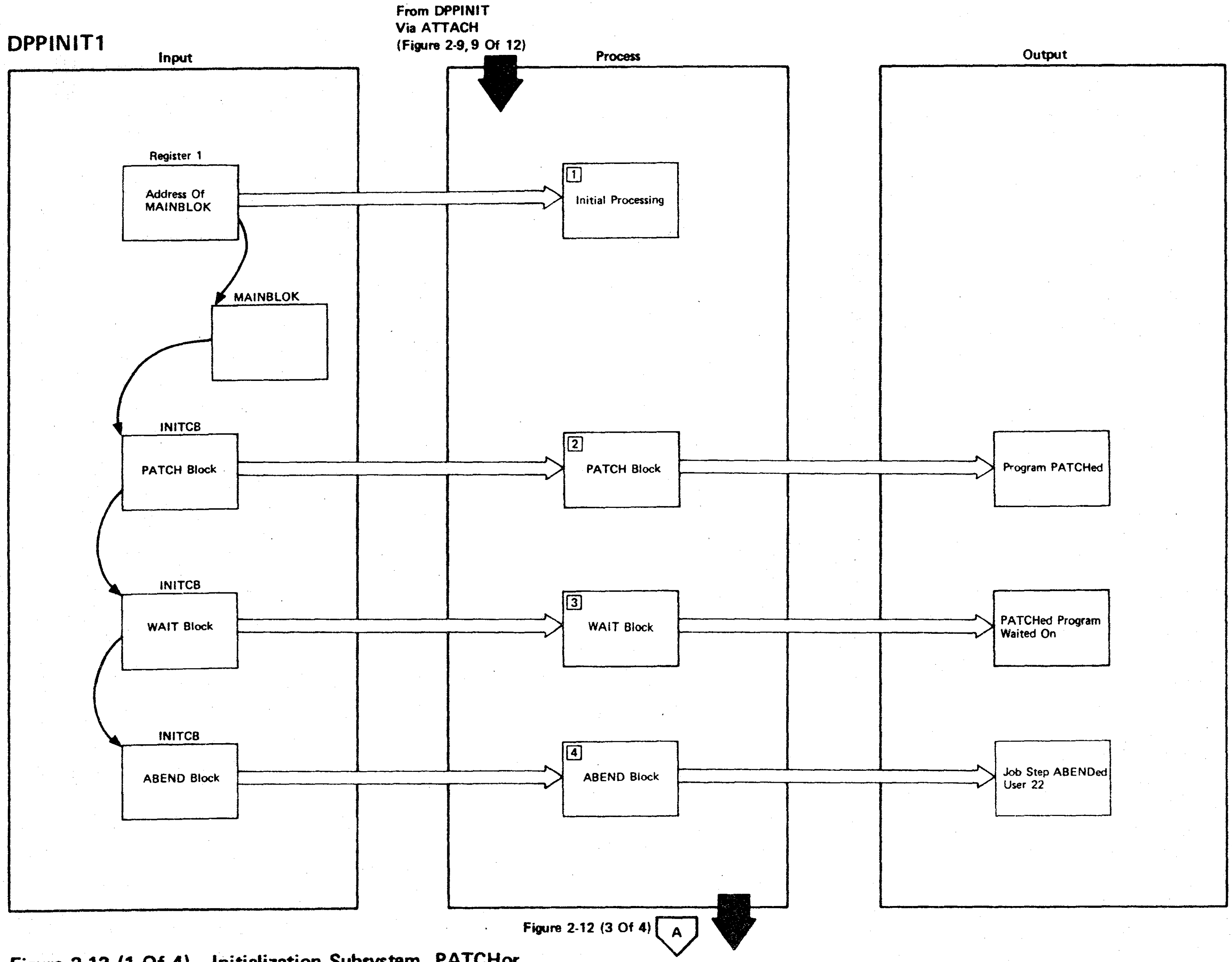


Figure 2-12 (1 Of 4) - Initialization Subsystem PATCHor

Figure 2-12 (2 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Register conventions, get the XCVT address. Issue message to indicate PATCH processing has begun.	DPP041I	DPPINIT1
2	Get the PROBL and SUPL addresses from INITCB and issue the PATCH. If PARAM= was coded, PATCH is issued with ECB option. Also if the PATCH precedes a RESTART statement, the PATCH is issued with ECB option; otherwise, no ECB option is specified. If PATCH return code is nonzero, ABEND the job step with a code 31.	USER 31	DPPINIT1
3	The INITWLST flag is checked to see if the WAIT is on a list of ECBs. If it is, the INITWTCT is obtained, and a WAIT is issued on the list. If no wait list, the WAIT is issued on a single ECB with the ECB being the INITECB field of the PATCH INITCB. When the ECB is posted, the completion code is checked, and if no error occurred processing continues. If an error (POST code nonzero) occurs, an error message is issued and the nonzero POST code is zerod. If the bad POST was for a PATCH prior to the RESTART, the task ABENDs with a code 35.	DPP044I USER 35	DPPINIT1
4	The program issues a STIMER WAIT for the specified time. When the time expires, the job step is ABENDed with a code 22.	USER 22	DPPINIT1

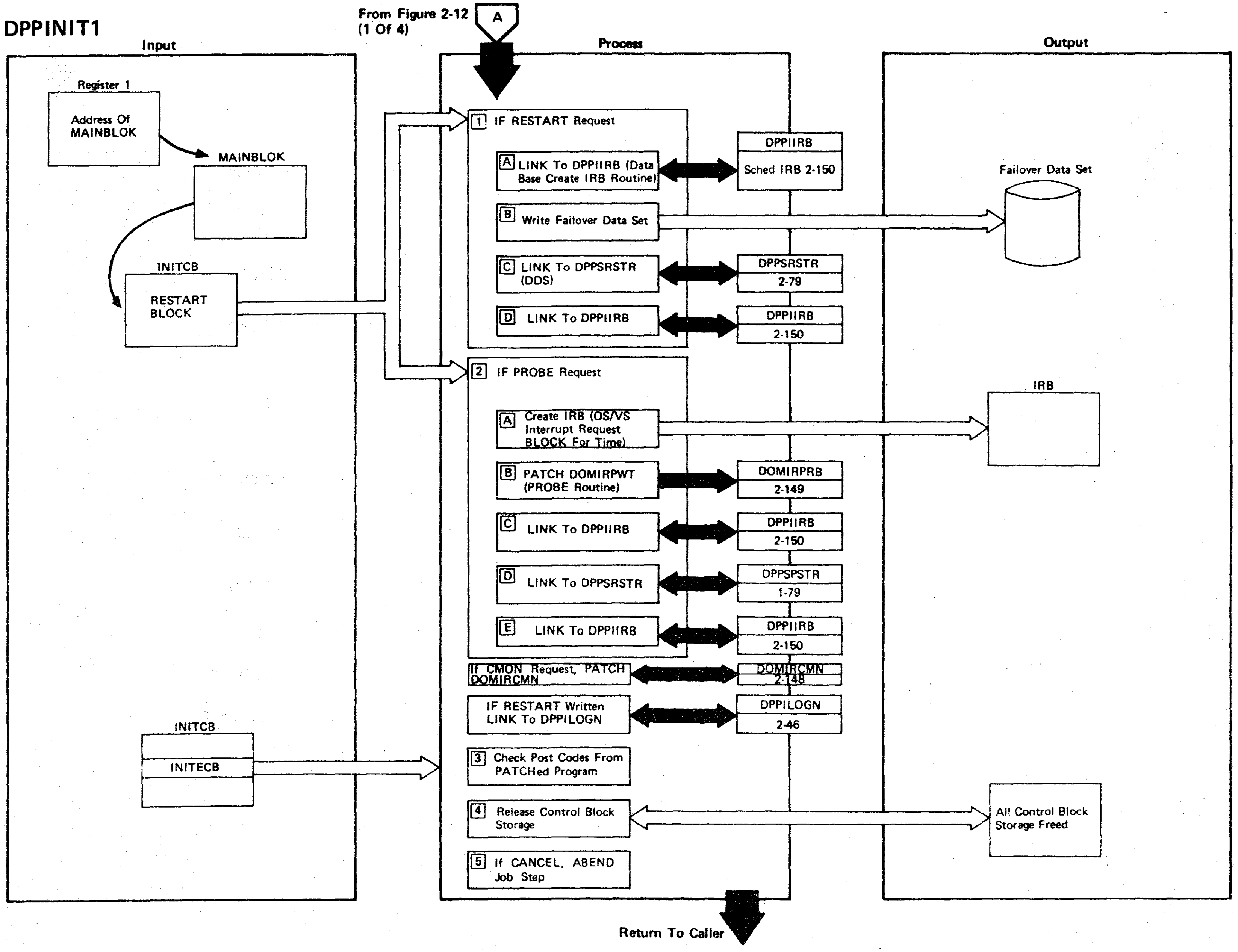


Figure 2-12 (3 Of 4) - Initialization Subsystem PATCHor

Figure 2-12 (4 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If RESTART request		DPPINIT1
	A Link to DPPIIRB (Data Base Create IRB Routine)		
	B A WAIT is issued on all previous PATCHes. The failover restart data set is then written and then the restart flags are propagated to all PATCHes following the RESTART block. The flags are stored in the PROBL. If it is a SLAVE partition that has been restarted, the WRITE RESTART is bypassed and an error message is issued.	DPP054I	
	C Link to DDS failover restart routine (DPPSRSTR)		
	D Link to DPPIIRB		
2	If PROBE request		DPPINIT1
	A Create IRB for the time function		
	B PATCH DOMIRPWT (PROBE Routine)		
	C Link to DPPIIRB		
	D Link to DPPSRSTR		
	E Link to DPPIIRB		
3	All ECBs are checked. Nonzero POST codes will cause an error message to be issued.	DPP044I	DPPINIT1
4	All control block storage is FREEMAINed.		DPPINIT1
5	ABEND job step with a code 45.	USER 45	DPPINIT1

DPPISTAE

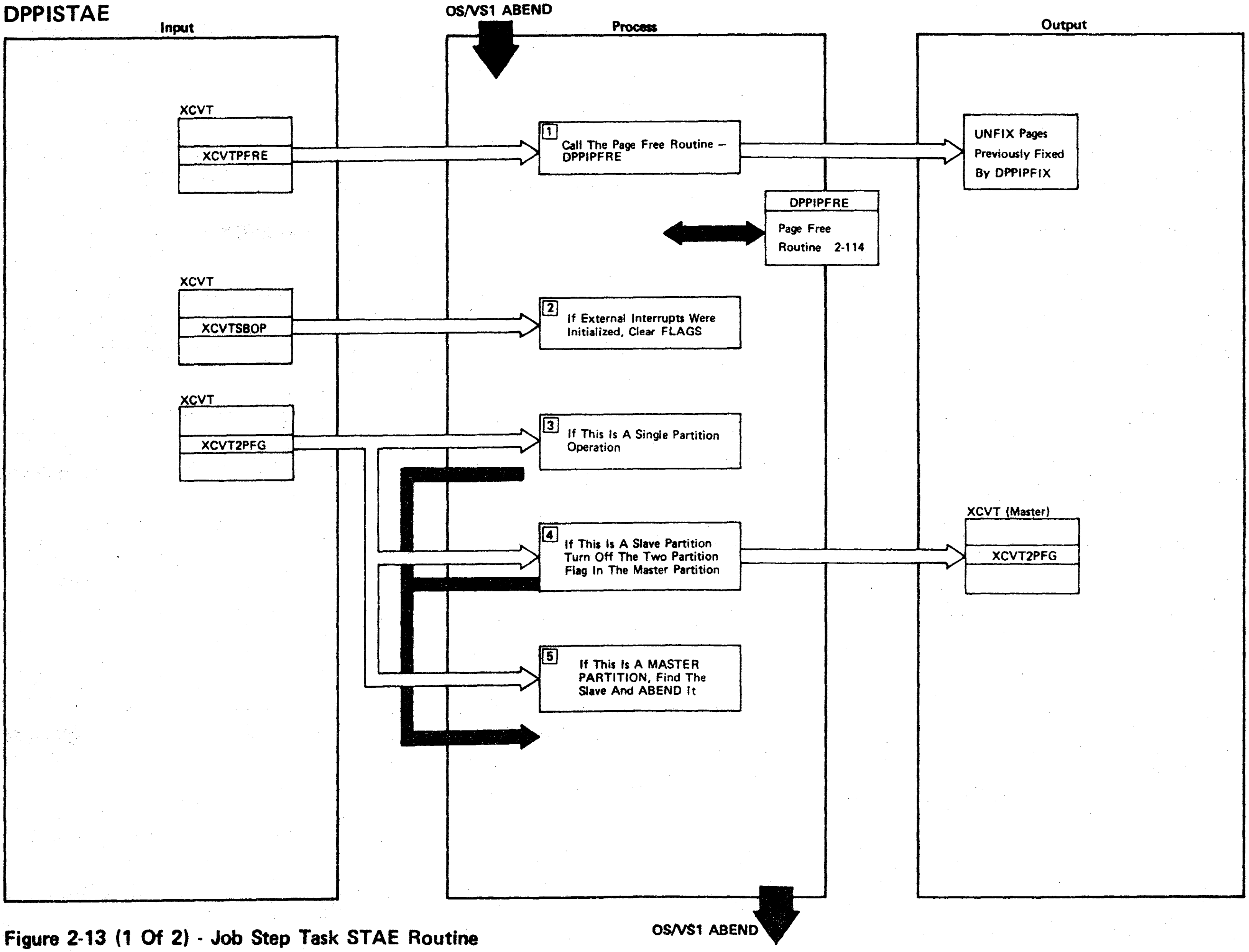

OS/VS1 ABEND
Input
Process
Output
XCVT
XCVTPFRE
1 Call The Page Free Routine – DPPIPFRE
UNFIX Pages Previously Fixed By DPPIPFIX
DPPIPFRE
Page Free Routine 2-114
XCVT
XCVTSBOP
2 If External Interrupts Were Initialized, Clear FLAGS
XCVT
XCVT2PFG
3 If This Is A Single Partition Operation
4 If This Is A Slave Partition Turn Off The Two Partition Flag In The Master Partition
XCVT (Master)
XCVT2PFG
5 If This Is A MASTER PARTITION, Find The Slave And ABEND It
OS/VS1 ABEND

Figure 2-13 (1 Of 2) - Job Step Task STAE Routine

Figure 2-13 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The page free (unfix) routine, DPPIPFRE, is branched to unfix any pages fixed by the initialization routine.		DPPISTAE
2	If external interrupts have been initialized at initialization time, the flags (XCVTSBOP) are reset.		DPPISTAE
3	If this is a single partition run, control is returned to ABEND with register 15 cleared to indicate no retry.		DPPISTAE
4	If this is a SLAVE partition in a two partition run, the MASTER partition's XCVT is found, and the two partition flag (XCVTF2PT) is turned off, and control returned to ABEND with register 15 zero to indicate no retry.		DPPISTAE
5	If this is a MASTER partition in a two partition run, the SLAVE partition's job step task TCB is located, and the SLAVE job is ABENDed with a code 41. Control is then returned to ABEND with register 15 zero to indicate no retry.	USER 41	DPPISTAE

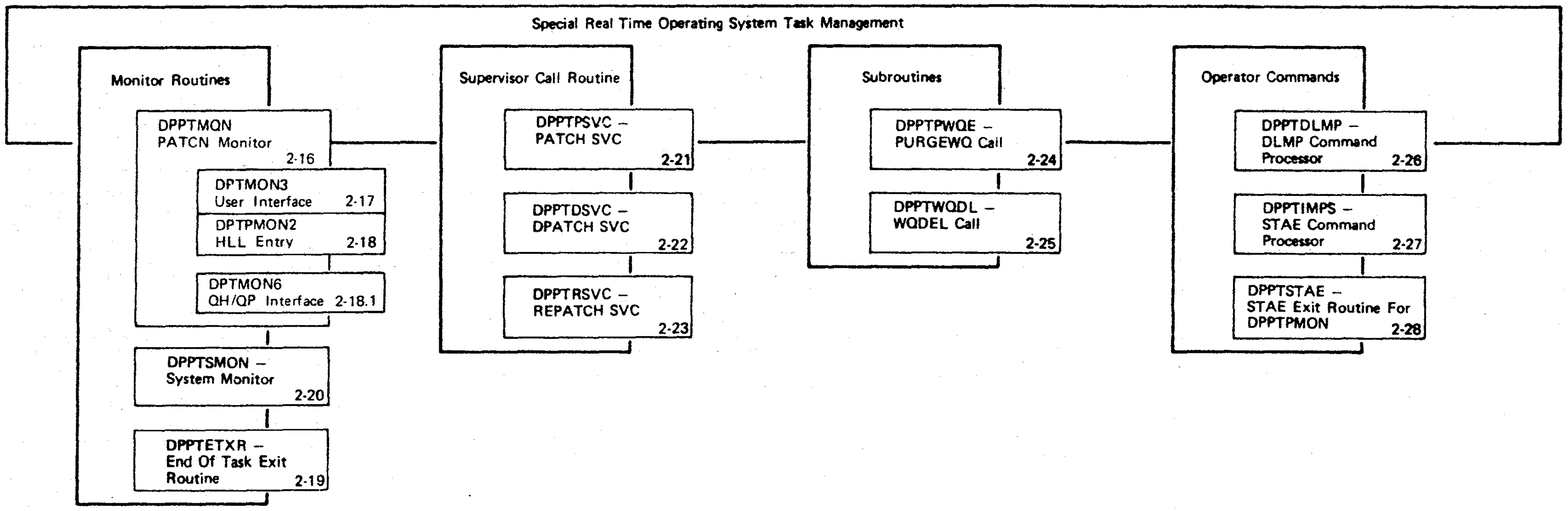


Figure 2-14 (1 of 2) Special Real Time Operating System Task Management Overview

Task Management

The Special Real Time Operating System's task management services are an extension of the OS/VS1 tasks supervisor to make more efficient use of systems resources in a real time processing system. These additional services are provided by the Special Real Time Operating System through the use of SVC routines, monitor routines, operator commands, and service subroutines as shown in Figure 2-14.

The PATCH monitor routine and the system monitor routine, DPPTPMON and DPPTSMON respectively, receive control from the Special Real Time Operating System initialization module, DPPINIT, and form the heart of task management. DPPTSMON executes under the job step task and performs the services required by the real time system as a whole (i.e., create new subtasks, LOAD reentrant modules, etc.). DPPTPMON executes under each subtask created by DPPTSMON and interfaces with the user routines as required on a PATCH macro call. The relationship between the user program and the task management routines is shown in Figure 2-15.

The task management routines provide most of the communication between partitions in a two-partition environment. This is done internally to each routine and does not affect the overall logic flow or the function of that routine.

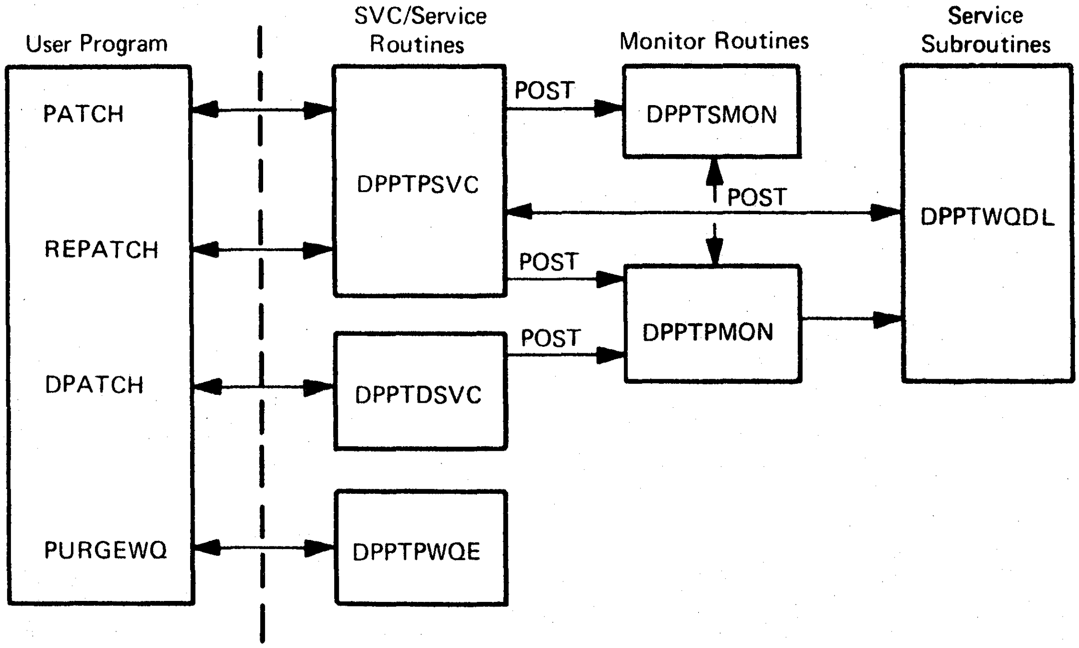


Figure 2-15. Task Management-User Program Relationships

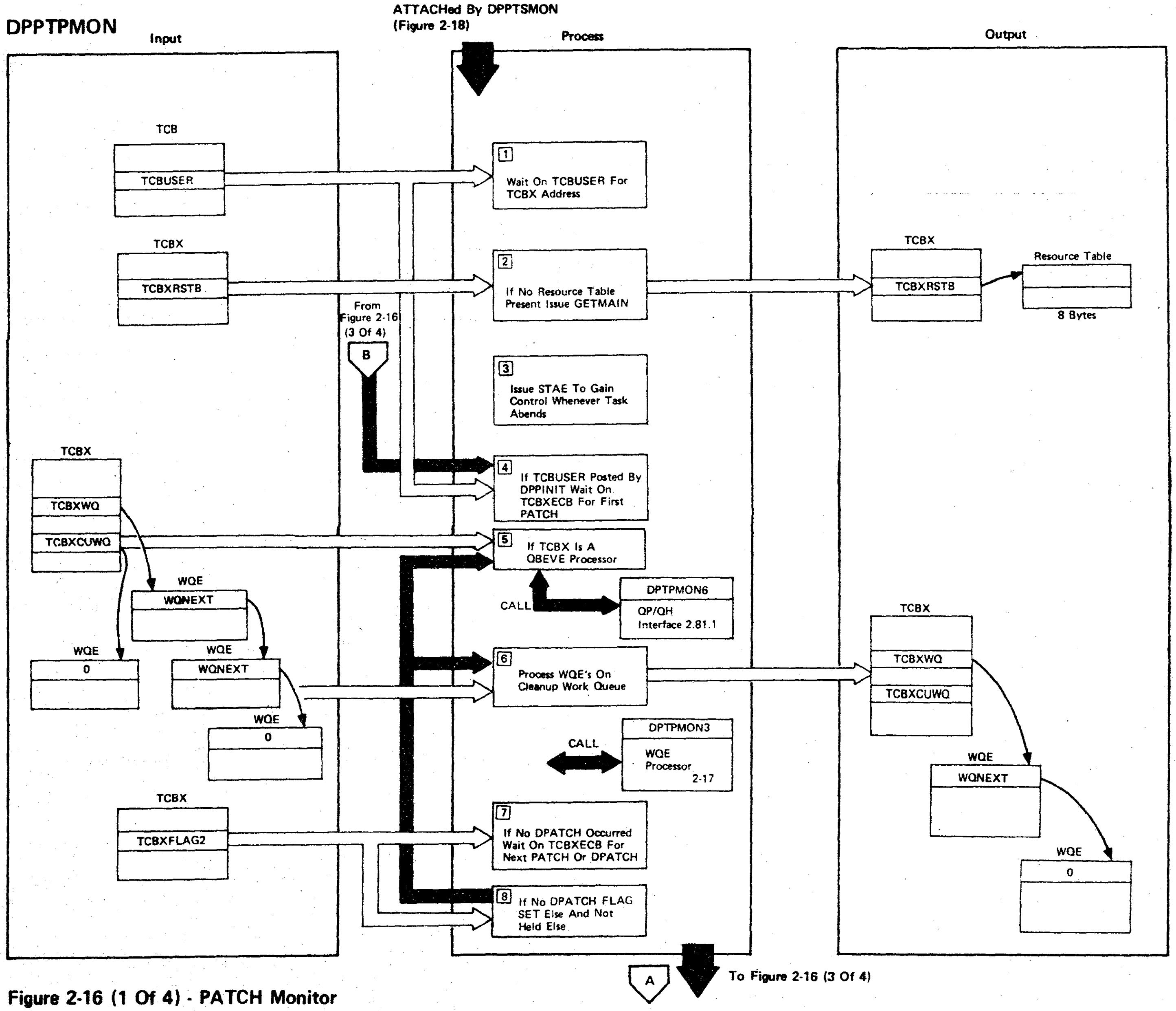

DPPTPMON
Input
ATTACHed By DPPTSMON
(Figure 2-18)
Process
Output
TCB
TCBUSER
1
Wait On TCBUSER For TCBX Address
TCBX
TCBXRSTB
2
If No Resource Table Present Issue GETMAIN
TCBX
TCBXRSTB
Resource Table
8 Bytes
From Figure 2-16 (3 Of 4)
B
3
Issue STAE To Gain Control Whenever Task Abends
TCBX
TCBXWQ
TCBXCUWQ
4
If TCBUSER Posted By DPPINIT Wait On TCBXECB For First PATCH
5
If TCBX Is A QBEVE Processor
CALL
DPTPMON6
QP/QH Interface 2.81.1
WQE
WQNEXT
WQE
0
WQE
WQNEXT
WQE
0
6
Process WQE's On Cleanup Work Queue
TCBX
TCBXWQ
TCBXCUWQ
DPTPMON3
CALL
WQE Processor 2-17
WQE
WQNEXT
WQE
0
TCBX
TCBXFLAG2
7
If No DPATCH Occurred Wait On TCBXECB For Next PATCH Or DPATCH
8
If No DPATCH FLAG SET Else And Not Held Else
A
To Figure 2-16 (3 Of 4)

Figure 2-16 (1 Of 4) - PATCH Monitor

Figure 2-16 (2 of 4)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The PATCH monitor is attached by DPPINIT during Special Real Time Operating System initialization or by the system monitor DPPTSMON thereafter. The address of the TCBX is put into the TCBUSER field via POST by the mother task.		DPPTPMON
2	It is checked if a resource table address is present in the TCBX and if not, a resource table plus work area are obtained through GETMAIN and the address stored into TCBXRSTB.		DPPTPMON
3	The STAE specifies DPPTSTAE as the exit routine.		DPPTPMON
4	If DPPINIT posted TCBUSER, this is an initial TCB on the FREE chain (TMCTFREE), and the PATCH monitor waits here on TCBXECB for the first PATCH.		DPPTPMON
5	If TCBX is a queue processor (QP), then segment DPTPMON6 is used to select work from one of the queue holders associated with this QP.		DPPTPMON
6	If any WQEs are on the cleanup work queue TCBXCUWQ they are dechained, DPPSCLUP is called and the WQE-DELETE routine is invoked through a branch entry to delete the WQE. Then the top WQE is dequeued from the TCBXWQ chain, it becomes the "current" WQE and its address is kept in TCBXCWQ. Each WQE is processed as long as WQEs are present on the queue and no DPATCH TYPE = U (unconditional) is received.		DPPTPMON
7	If no DPATCH occurred, indicating that the queue is empty, the PATCH monitor waits here for a next PATCH or DPATCH.		DPPTPMON
8	If no DPATCH flag is set (TCBXFLG2) and not HELD (TCBFLG3), control goes back to step 5 for processing of the received PATCH. Otherwise, if a DPATCH was received, control goes to A. (Figure 2-16 (3 of 4)). If task is being HELD, the PATCH monitor WAITs until released before processing additional work queues.		DPPTPMON

DPPTPMON

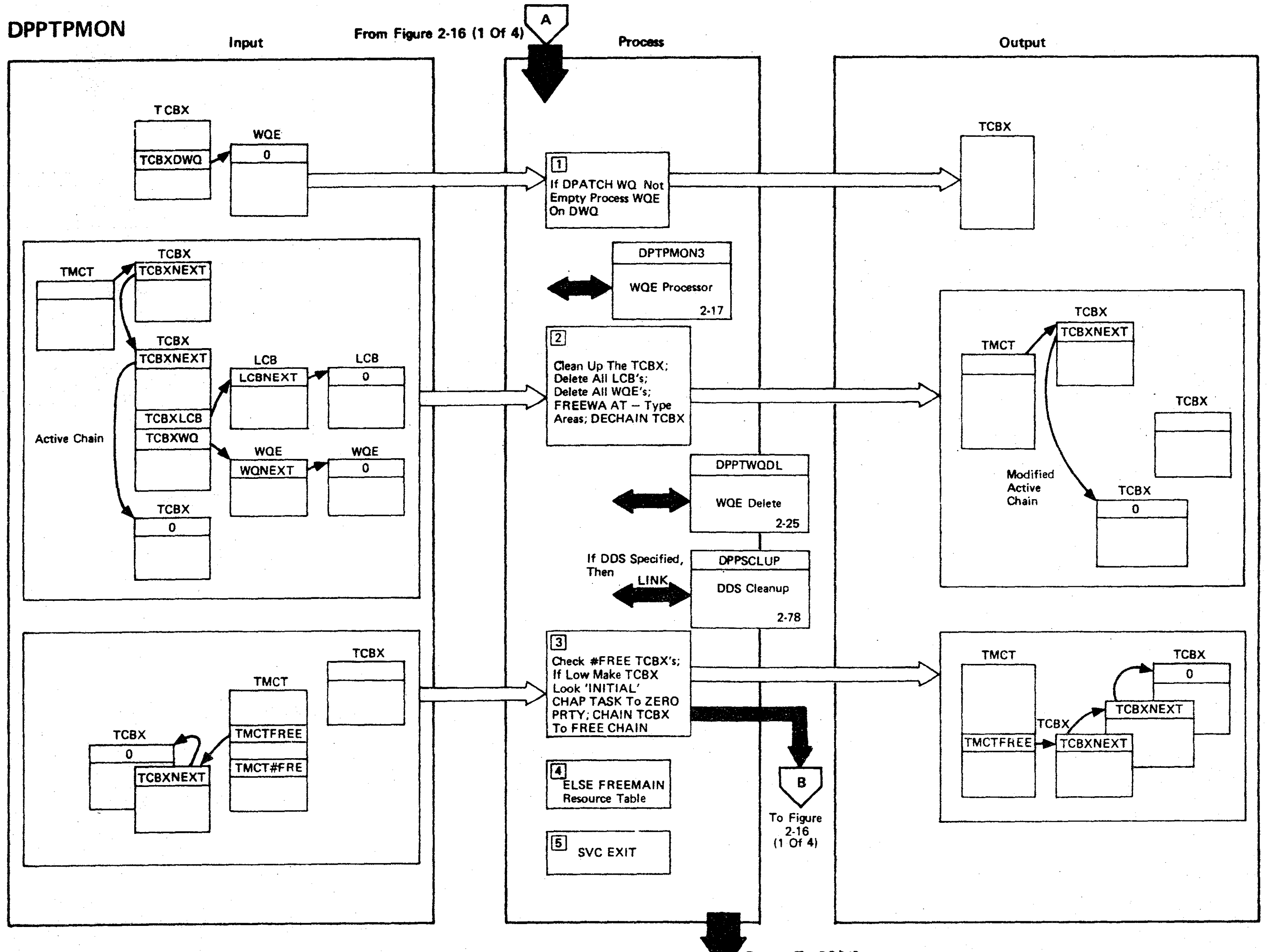


Figure 2-16 (3 Of 4) - PATCH Monitor

Figure 2-16 (4 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the DPATCH work queue TCBXDWQ is not empty, that WQE is dechained and its address is kept in TCBXCWQ. The WQE is processed like any other WQE.		DPPTPMON
2	The TCBX will then be cleaned up. DPPSCLUP is called to clean up DDS. Remaining WQEs are deleted by using the WQDL routine. AT-Type GETWA areas are freed using the special entry to FREEWA. Remaining LCBs are deleted. If there is a corresponding LCB on the TMCT-LCB chain, that LCB's use-count is decremented. If it goes to zero, the flags LCBFDEL and TMCTLCBD are set to cause DPPTSMON to delete the program. If the program was not reentrant, it is deleted here. Then the TCBX is dechained from its active independent or dependent task chain, TMCTAIND or TMCTADEP, respectively.	DPP016I	DPTPMON1
3	A check is made for the number of TCBXs on the TMCTFREE chain. If it is low, the TCBX is further cleaned up to look "initial" (TCBXNAME, TCBXPARM, TCBXFLGs), the task is CHAP'ed down to zero priority, and the TCBX is chained to the FREE chain. Control now goes back to step 4 of Figure 2-16 (1 of 4), where the PATCH monitor will wait for a new "first" PATCH.		DPTPMON1
4	If the limit number of free TCBXs is already reached, the flag TCBX1TRM is set, and FREEMAIN of resource table plus work area is done. Then SVC EXIT is issued to terminate the task.		DPPTPMON

DPTPMON3

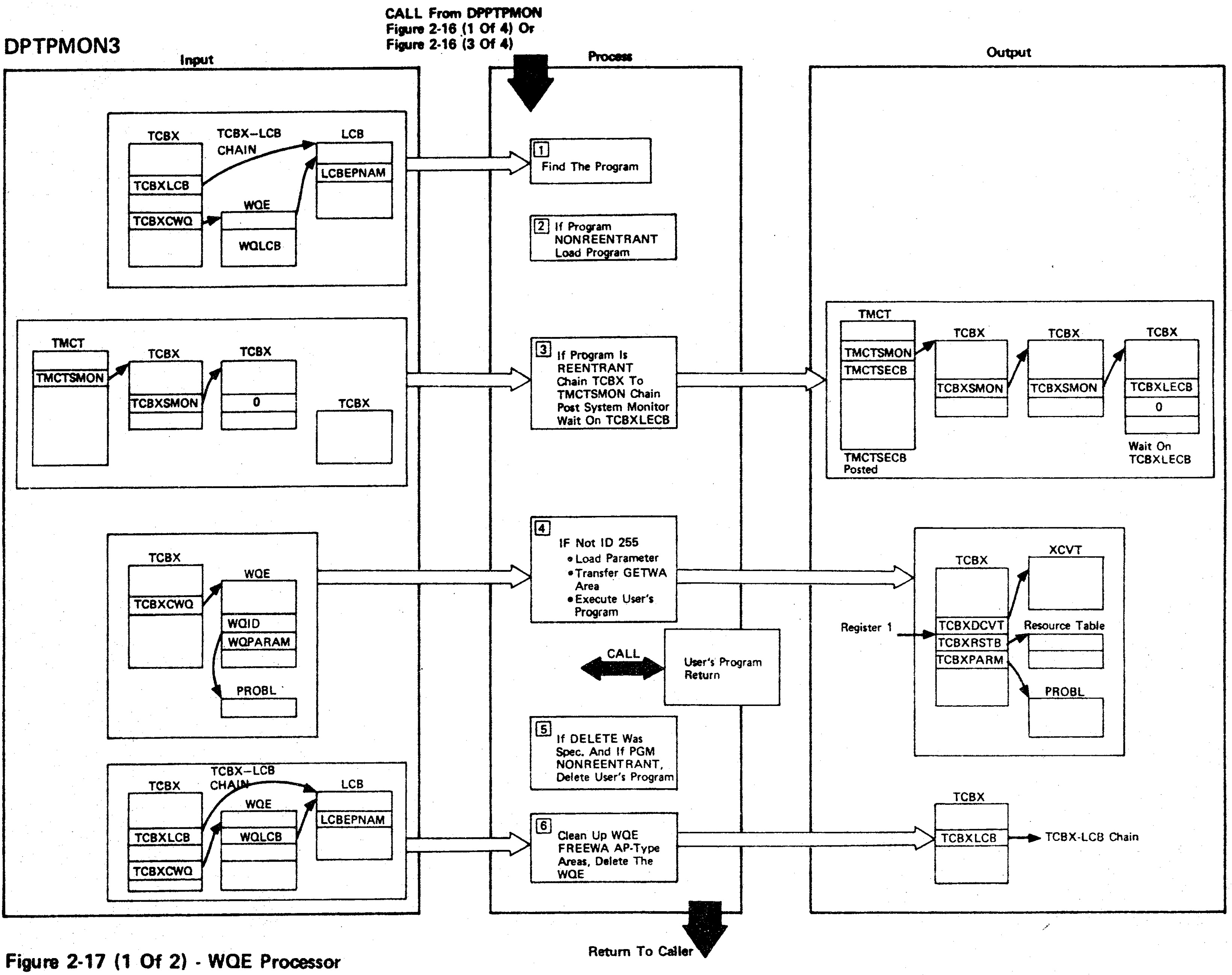
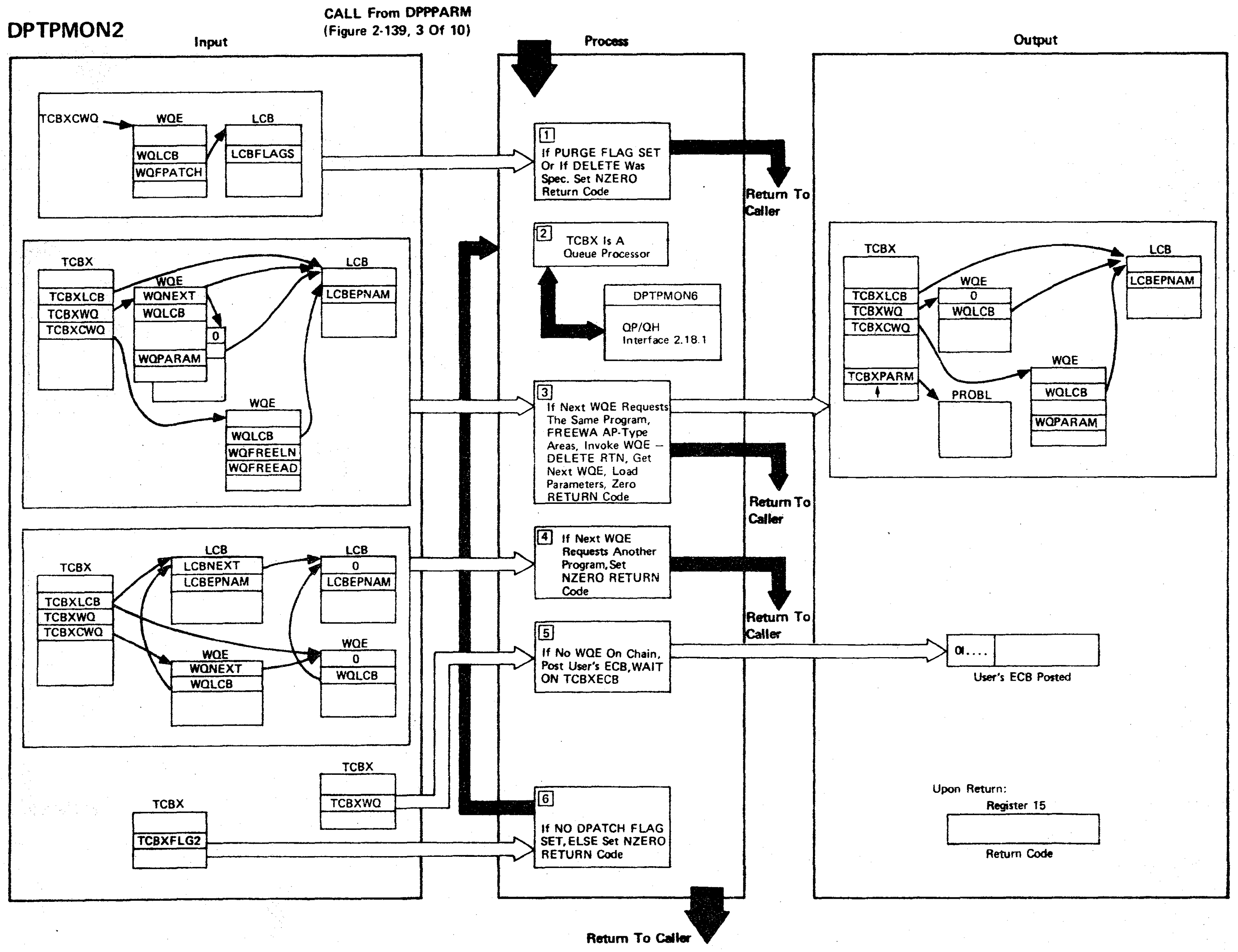


Figure 2-17 (1 Of 2) - WQE Processor

Figure 2-17 (2 of 2)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the purge flag is set in the LCB pointed to by the current WQE, DPPTPMON waits for dynamic load module PURGE to complete. If the LCB is unresolved, a search is made to find the program on the TMCT-LCB chain. If found, the TCBX-LCB is pointed to the TMCT-LCB, the user count is incremented, and the EP address is copied. If not found on the chain, a BLDL is issued to locate the program.	DPP014I DPP015I	DPTPMON3
2	If the program is non-reentrant, it is LOADed and its EP address kept in the LCB. If it is a Queue Processor task build a duplicate LCB for this Q Proccessor Reentrant programs are task oriented and the CB for the QH is not.		DPTPMON3 DPTPMON3
3	If the program is reentrant, flags LCBFLOAD and TCBX1LCB are set, and the TCBX is chained to the TMCTSMON chain. Then the system monitor is POSTed (TMCTSECB) and the PATCH monitor waits on TCBXLECB.		DPTPMON4
4	The address of the PROBL is stored into TCBXPARM. If ID is not 255, the address of TCBXDCVT is loaded into register 1 and the user's program is given control via BALR 14, 15.		DPTPMON4
5	The user's program will return here. If the program is nonreusable or if it is reusable and DEL was specified, it is DELETEd.		DPTPMON5
6	If the purge flag is set and an ECB address was supplied by DPPTDLMP, the ECB is POSTed. If any AP-type GETWA area is chained to TCBXOFWD, FREEWA is executed (via the branch entry of FREEWA.) Then the WQDL routine is invoked via branch entry to delete the WQE.		



1 If PURGE FLAG SET Or If DELETE Was Spec. Set NZERO Return Code

2 TCBX Is A Queue Processor

3 If Next WQE Requests The Same Program, FREEWA AP-Type Areas, Invoke WQE – DELETE RTN, Get Next WQE, Load Parameters, Zero RETURN Code

4 If Next WQE Requests Another Program, Set NZERO RETURN Code

5 If No WQE On Chain, Post User's ECB, WAIT ON TCBXECB

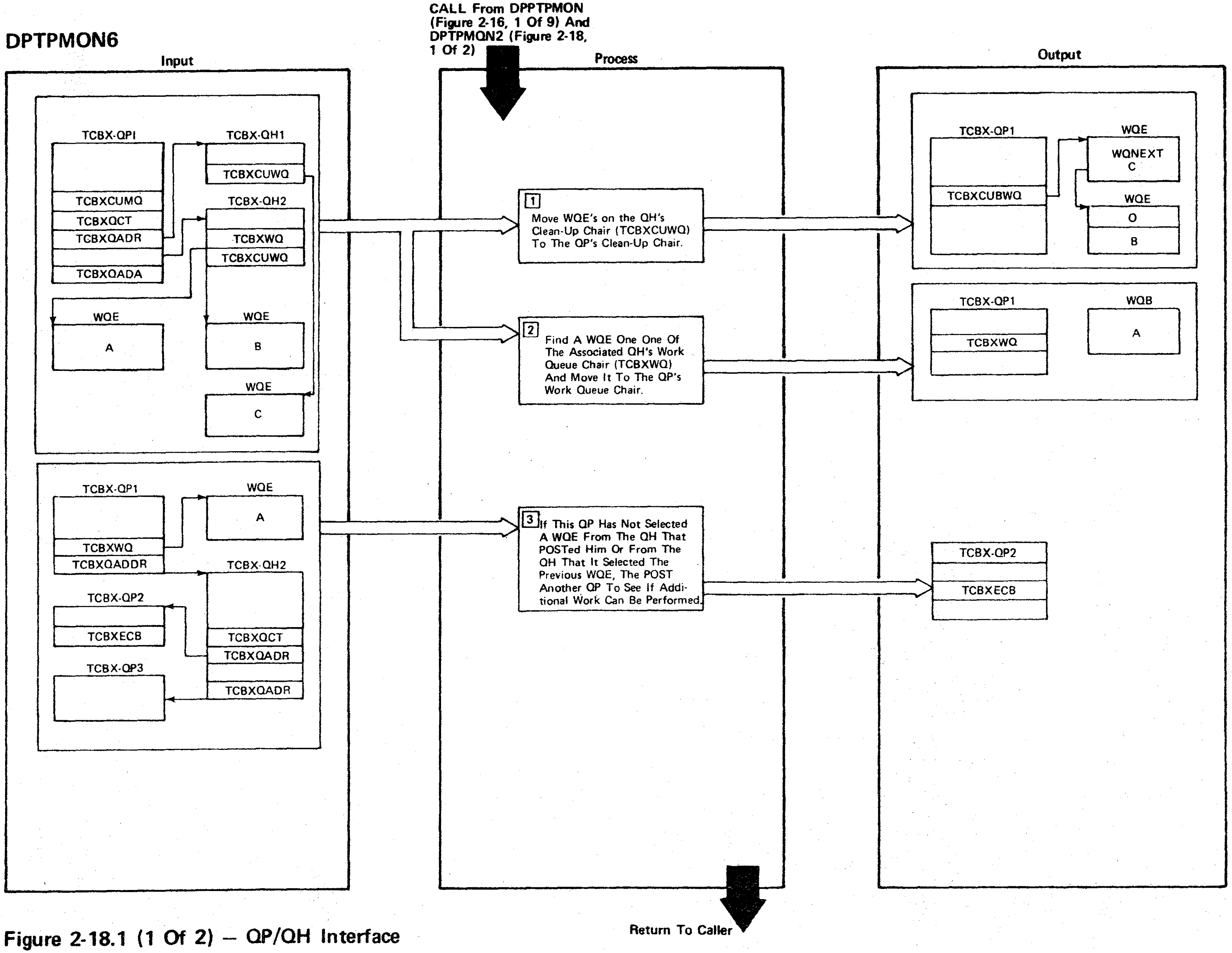
6 If NO DPATCH FLAG SET, ELSE Set NZERO RETURN Code

Figure 2-18 (1 Of 2) - PATCH Monitor - High Level Language Entry

Figure 2-18 (2 Of 2)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
	This part of the Patch Monitor is entered only from the high level language interface programs (DPPPARM for PL/I or DPPFPRM for FORTRAN).		
1	If the purge flag is set or if DEL was specified, return to caller with a nonzero return code.		DPTPMON2
2	If TCBX is a queue processor (QP) then segment DPTPMQN6 is used to select work from one of the queue holders associated with this QP.		DPTPMON2
3	A check is made if the next WQE requests the same program. If yes, the old WQE is cleaned up; AP-Type GETWA areas are freed and the old WQE deleted via branch entry to the WQDL Routine. Then the new WQE is scheduled, and the PROBL address loaded into TCBXPARM. If ID is not 255, the return code is set to zero and control returned to the caller. If ID is 255, no return is performed, but the routine continues to check the next WQE while WQEs are present on TCBXWQ and no DPATCH flag is set.		DPTPMON2
4	If the next WQE requests a different program, the current control is passed back to the caller with a nonzero return code.		DPTPMON2
5	If no WQE is on TCBXWQ, the user's ECB is posted to indicate that processing of this work queue is completed, the ECB address is cleared from the WQE, and the Patch Monitor waits on TCBXECB for a next PATCH or DPATCH.		DPTPMON2
6	After TCBXECB is posted and if no DPATCH flag is set in TCBXFLG2, the routine continues processing with step 2. If a DPATCH occurred, the return code is set nonzero and control returns to the caller.		DPTPMON2

DPTPMON6



1 Move WQE's on the QH's Clean-Up Chair (TCBXCUWQ) To The QP's Clean-Up Chair.

2 Find A WQE One One Of The Associated QH's Work Queue Chair (TCBXWQ) And Move It To The QP's Work Queue Chair.

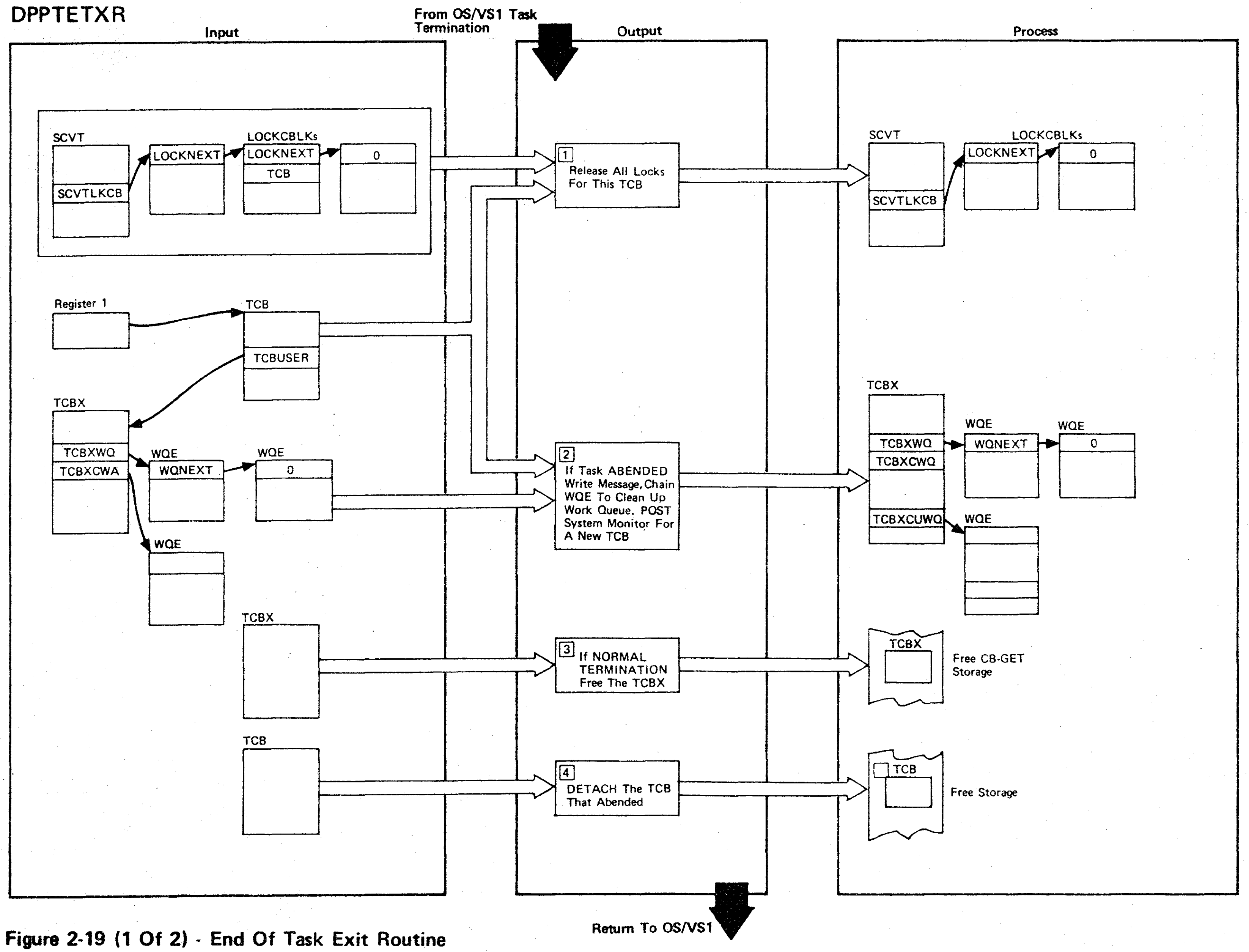
3 If This QP Has Not Selected A WQE From The QH That POSTed Him Or From The QH That It Selected The Previous WQE, The POST Another QP To See If Additional Work Can Be Performed.

Figure 2-18.1 (1 Of 2) — QP/QH Interface

Figure 2-18.1 (2 Of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The address of each associated QH TCBX is contained in the TCBX for that QP. Since a QH is not associated to an QS task the clean-up work queue for the WHs are moved to the TCBX for the QP.		DPTMONG
2	The chain of QHs is searched looking for and available work queue (is a work queue on the work queue chain of a QH the QH is not HELD, and the QH is not sequential with another work queue currently being processed by another QP).		DPTPMONG
3	The chain of QPs associated with the QH that the work queue was selected from is searched looking for a available QP (i.e. a dormant QP that is not HELD and has not been previously posted).		

DPPTETXR



1 Release All Locks For This TCB

2 If Task ABENDED Write Message, Chain WQE To Clean Up Work Queue. POST System Monitor For A New TCB

3 If NORMAL TERMINATION Free The TCBX

4 DETACH The TCB That Abended

Figure 2-19 (1 Of 2) - End Of Task Exit Routine

Figure 2-19 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
	The End of Task Exit routine is specified when the PATCH monitor is attached by the Special Real Time Operating System initialization or the system monitor and it executes as an asynchronous exit routine of OS/VS1 task termination.		
1	The SCVTLKCB chain is searched for any LOCKCBLK referring to the ABENDing TCB and if found, an UNLOCK is issued. If it is not a daughter of the job step task, ABEND with a code 64.	USER 64	DPPTETXR
2	If the task ABENDed (TCBCMP nonzero), a message is issued. If flag TCBX1TRM is not set, the WQE is chained to the TCBXCUWQ cleanup work queue. The flags TCBX1TCB and TCBX1CHP are set, and the TCBX is chained to the system monitors request chain TMCTSMON. Then the system monitor is posted (TMCTSECB).	DPP010I DPP011I DPP012I DPP013I DPP018I	DPPTETXR
3	If the task terminated normally (TCBX1TRM is set), the TCBX is freed.		DPPTETXR
4	The ABENDing task's TCB is detached to remove it from the OS/VS1 TCB chains and release its storage from fixed PQA. Then the routine returns to OS/VS1.		DPPTETXR

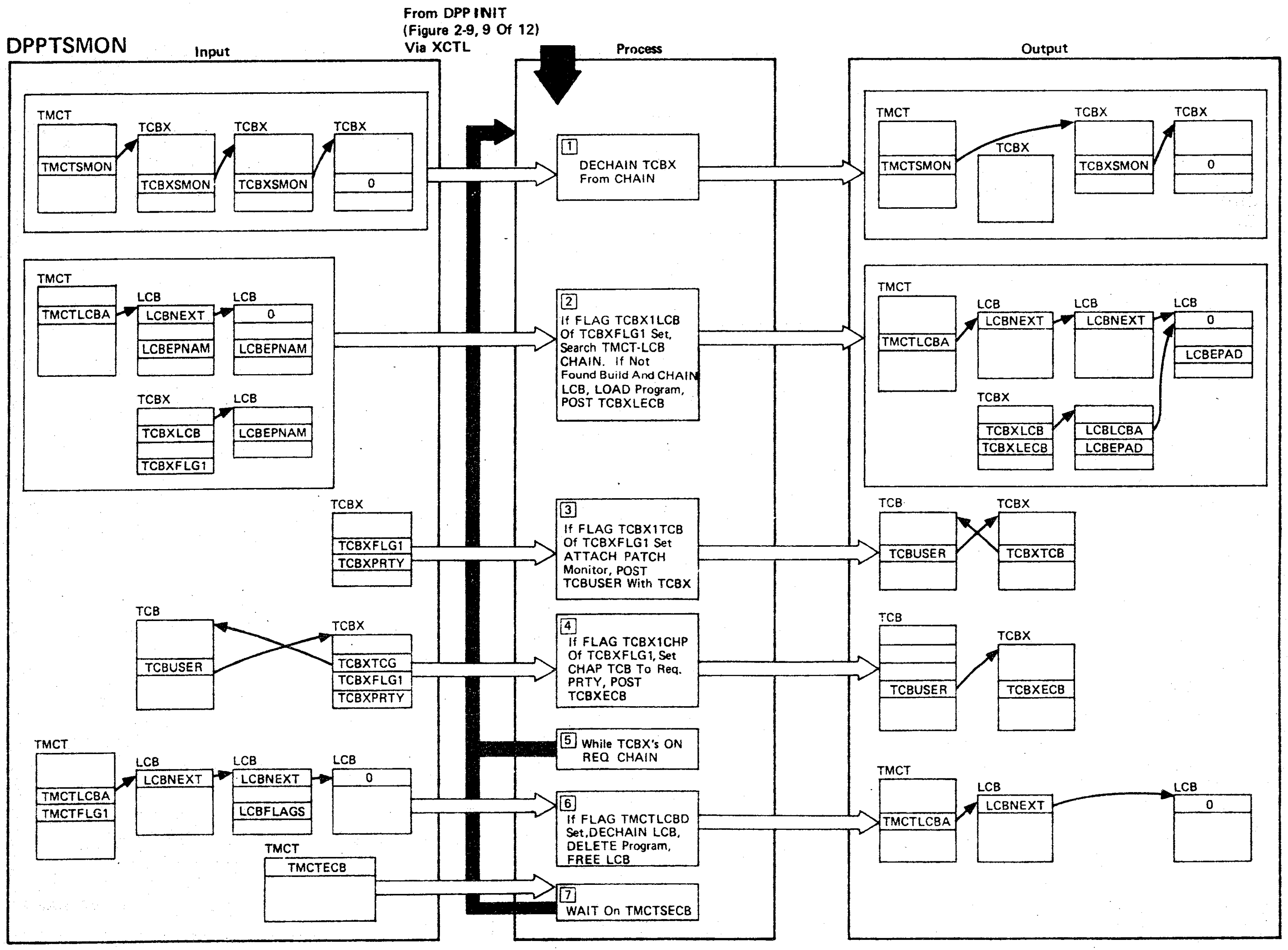


Figure 2-20 (1 Of 4) - System Monitor

Figure 2-20 (2 Of 4)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
	The System Monitor is entered via XCTL from the Special Real Time Operating System initialization and executes under the job step task TCB in a never ending loop as long as the real time system is running.		
1	The first TCBX is dechained from the TMCTSMON chain, the system monitors request chain, and the flag byte TCBXFLG1 is inspected for the kind of service requested.		DPPTSMON
2	If flag TCBX1LCB is set, the TMCT-LCB chain is searched for a program with the same name. If found, the TCBX-LCB is pointed to the TMCT-LCB, the EP address is copied, and the use count is updated. If not found, a new LCB is built from CB-GET storage and chained, the program is loaded, and the EP address stored in both LCBS. However, if CB-GET storage for a new LCB is not available, the program is treated nonreentrant, loaded and its EP address stored in the TCBX-LCB only. Then the waiting PATCH monitor is posted (TCBXLECB).	DPP017I	DPTSMON1
3	If flag TCBX1TCB is set, a new patch monitor is attached with the specified priority, the TCB address is stored into the TCBX, and the TCBUSER field of the TCB is posted with the TCBX address.		DPPTSMON
4	If flag TCBX1CHP is set, the requesting task is CHAPed to the proper priority, and the waiting patch monitor is posted (TCBXECB).		DPPTSMON
5	While more TCBXs are chained to TMCTSMON, the system monitor continues to service these requests (step 1 above).		DPPTSMON

Figure 2-20 (3 Of 4)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
6	If flag TMCTLCBD is set, the TMCT-LCB chain is searched for LCBs that are requesting a DELETE service (LCBFDEL). The LCBs are dechained, the programs deleted and the LCBs are freed. If any LCB had the purge flag set, DPPTDLMP is posted.		DPPTSMON
7	After the system monitor has serviced all requests, he waits on TMCTSECB, and a POST for further service will pass control back to step 1.		DPPTSMON

Intentionally Blank

Figure 2-20 (4 of 4)

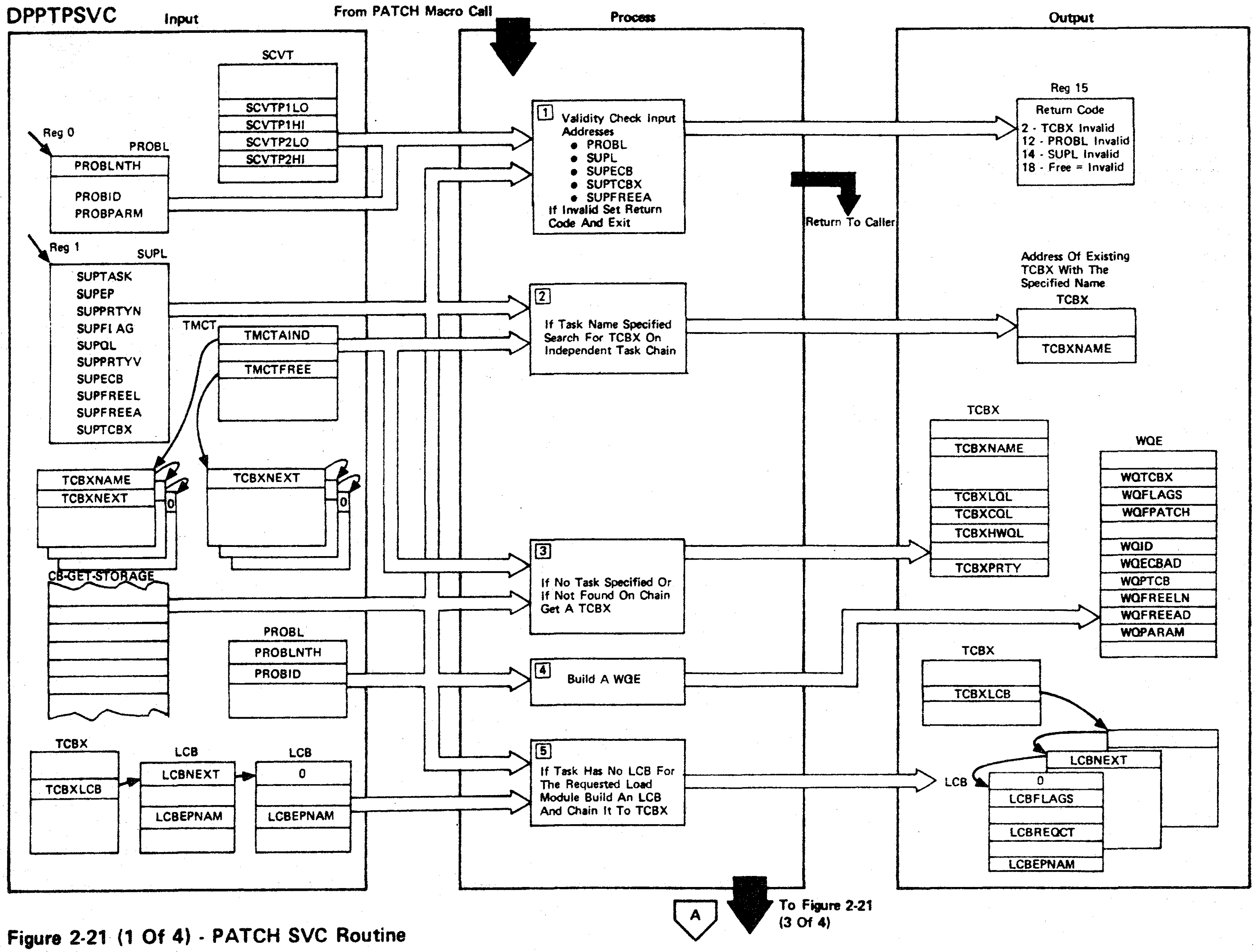
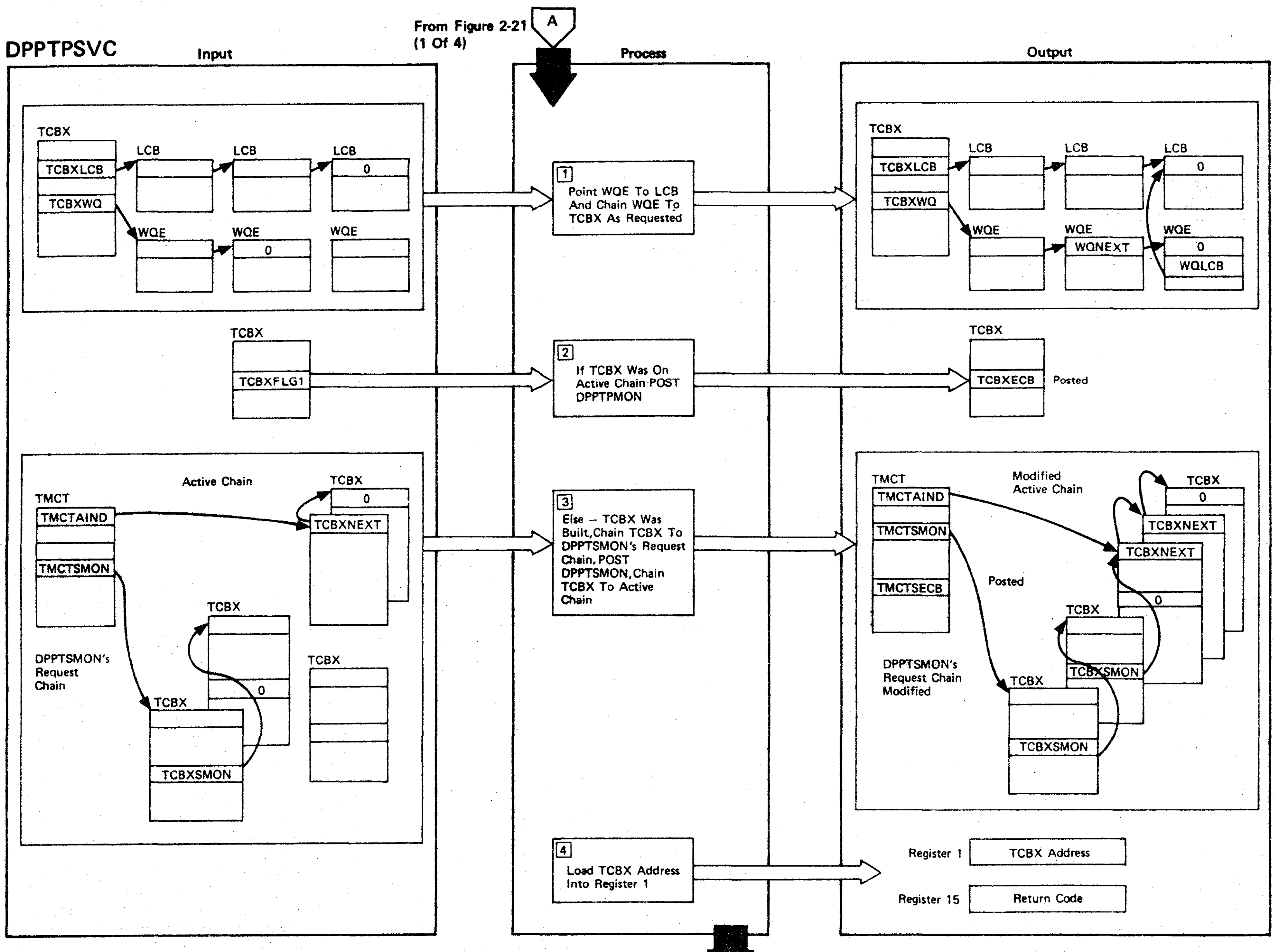


Figure 2-21 (1 Of 4) - PATCH SVC Routine

Figure 2-21 (2 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The Problem Parameter List (PROBL) and Supervisor Parameter List (SUPL) addresses passed to PATCH are checked, both must be nonzero. ECB, TCBX, and FREE addresses may be specified; if so, the specified address(es) are also checked. The addresses must be within the partition in a single partition environment or within either the MASTER or SLAVE partition in a two-partition environment.		DPTPSVC1
2	If a task name was specified, the PATCH is for an independent task. The independent task chain (TMCTAIND) is searched for the name given.		DPPTPSVC
3	If no task name was specified (the PATCH is for a dependent task), or if a TCBX with the given name does not exist, a free TCBX is taken from the FREE chain (TMCTFREE) or if none is available, CB-GET storage is obtained and a new TCBX is built. Transfer GETWA area if required.		DPTPSVC4
4	A work queue element (WQE) is built from CB-GET storage.		DPTPSVC3
5	The TCBXLCB chain is searched for an LCB with the given EP name. If none is found, an LCB is built from CB-GET storage and chained to the TCBX.		DPTPSVC3



1 Point WQE To LCB And Chain WQE To TCBX As Requested

2 If TCBX Was On Active Chain POST DPPTPMON

3 Else – TCBX Was Built, Chain TCBX To DPPTSMON's Request Chain, POST DPPTSMON, Chain TCBX To Active Chain

4 Load TCBX Address Into Register 1

Return To Caller

Figure 2-21 (3 Of 4) - PATCH SVC Routine

Figure 2-21 (4 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Point the WQE to the LCB and chain the WQE to the TCBX as requested in the QPOS operand of PATCH. LAST - chain to the end of the TCBXWQ chain FIRST - chain at the top of the TCBXWQ chain. In this case, if the limit queue length is already reached, the bottom WQE is dechained and chained to the cleanup work queue TCBXCUWQ instead.. DPATCH - chain this WQE to TCBXDWQ (one WQE only can be chained to the DPATCH work queue). QPQS-DPATCH is invaluable for queue holders and queue processors.		DPTPSVC3
2	If the TCBX was on the active chain (flag TCBX1CHP in TCBX is zero, for no CHAP is necessary in this case), the Patch monitor DPPTPMON is posted (TCBXECB). For PATCH is to queue holders the first inactive available queue processor for that queue holders is posted.		DPTPSVC3
3	Otherwise the TCBX is chained to the system monitor DPPTSMON's request chain (TMCTSMON - TCBXSMON), and DPPTSMON is posted (TMCTSECB).		DPTPSVC4
	Also, the TCBX is chained to the top of the proper active chain in the TMCT. TMCTAIND - if task name specified TMCTADEP - if no task name given		DPPTPSVC
	The DPATCH=W flag TCBX2DPW in the TCBX is set also in case it is a dependent task to stop processing in DPPTPMON upon completion of this work request.		
4	The return code is loaded into register 15, and if it is less than or equal to 8, the TCBX address is loaded into register 1; otherwise, it is cleared. Then the routine returns to the caller.		DPPTPSVC

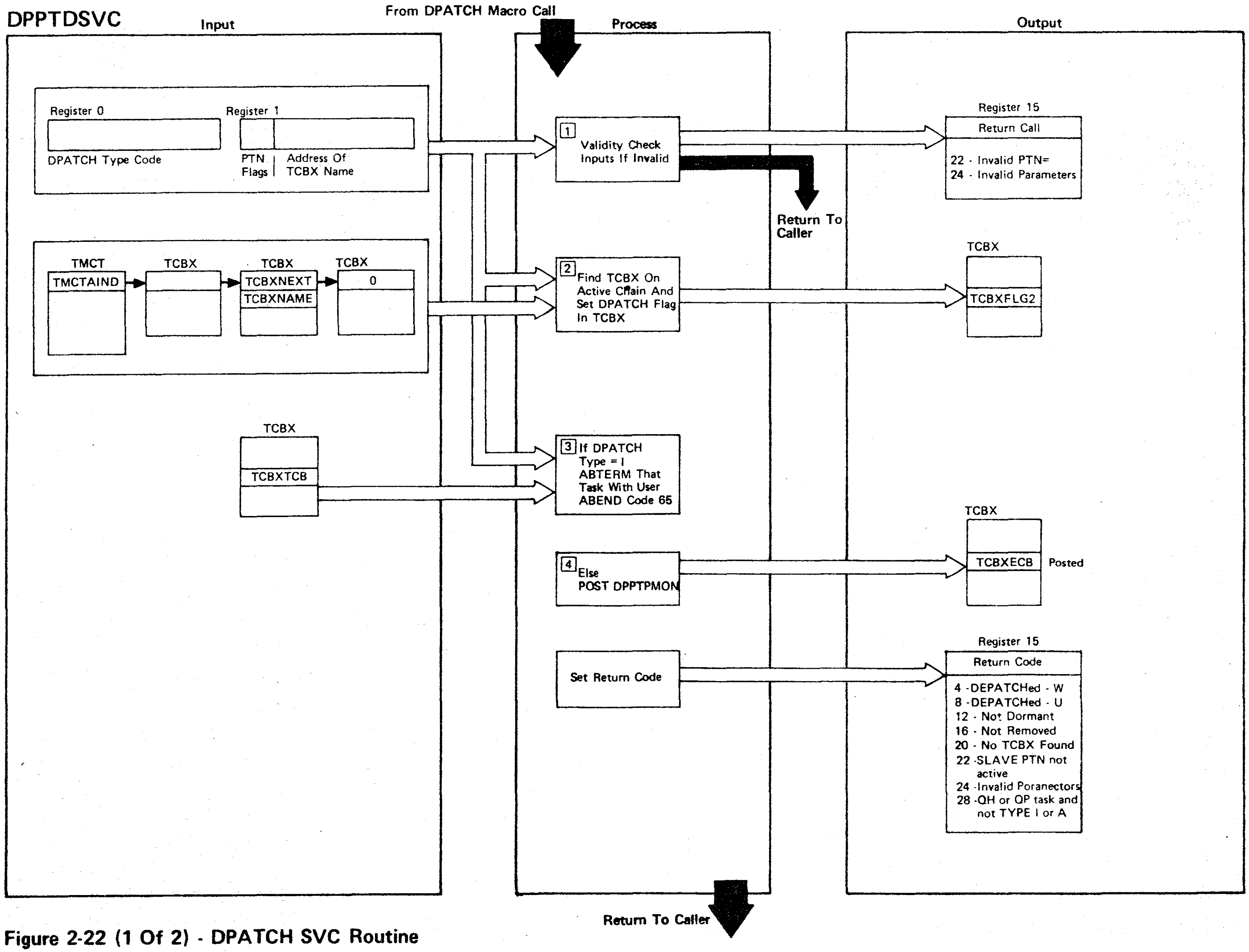


Figure 2-22 (1 Of 2) - DPATCH SVC Routine

Figure 2-22 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The contents of register 0 and 1 are checked. If register 1 is zero, the DPATCH is for the issuing task itself, in this case the TCBUSER field is checked. It must be within partition boundaries. If register 1 is nonzero, its content is the address of a storage field with the TCBXNAME that is to be DPATCHed. The address must be within partition boundaries. Register 0 must contain a valid TYPE code (0, 4, 8 or 12 corresponding to Type U, C, W, A, or I). Note: All type DPATCHs to queue holders are invalid and only DPATCH type A or I is valid for queue processors.		DPTDSVC1
2	If register 1 is nonzero, the TMCTAIND active task chain is searched for a TCBX with the specified name. A return code is loaded into register 15 if it cannot be found. The DPATCH - Flag corresponding to the DPATCH TYPE is set in the TCBX. If the same or another flag was already set, a return code in register 15 will indicate this.		DPTDSVC1
3	If DPATCH TYPE = I (immediate) was specified, the OS/VS1 ABTERM routine is invoked through a branch entry to ABTERM that task with a USER ABEND code of 65.	USER 65	DPPTDSVC
4	Otherwise DPPTPMON is posted (TCBXECB). The DPATCH SVC routine returns to the caller with a return code in register 15.		DPPTDSVC

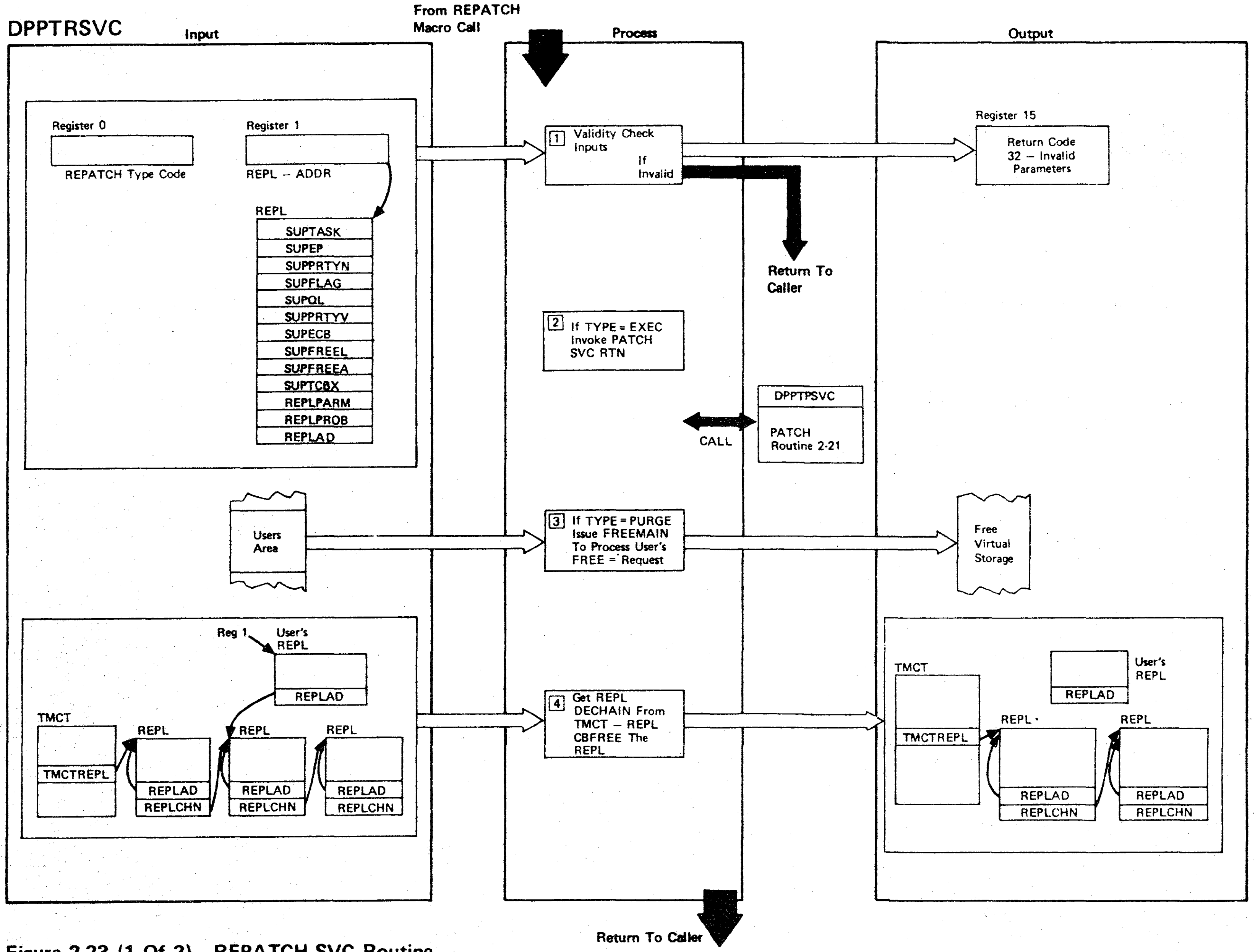


Figure 2-23 (1 Of 2) - REPATCH SVC Routine

Figure 2-23 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The contents of register 0 and 1 are checked. Register 0 must be 0 or 1, and register 1 must be a valid address of a REPL. Addresses are checked against partition boundaries of the own partition and if outside and two-partition operation boundaries also. If invalid, a return code of 32 is loaded into register 15, and the routine returns to the caller.		DPPTRSVC
2	If register 0 is zero (TYPE=EXEC), the input registers for the PATCH SVC routine are set up, and DPPTPSVC is invoked via branch entry. Any return code received upon return will be in turn passed to the caller of REPATCH.		DPPTRSVC
3	If register 0 is 1 (TYPE=PURGE) and a FREE= request was specified on the original PATCH, the FREEMAIN is issued.		DPPTRSVC
4	The address of the Special Real Time Operating System - supplied REPL is obtained and the REPL is dechained from the TMCT - REPL chain and freed.		DPPTRSVC
	The REPATCH SVC routine returns to the caller with a return code in register 15.		

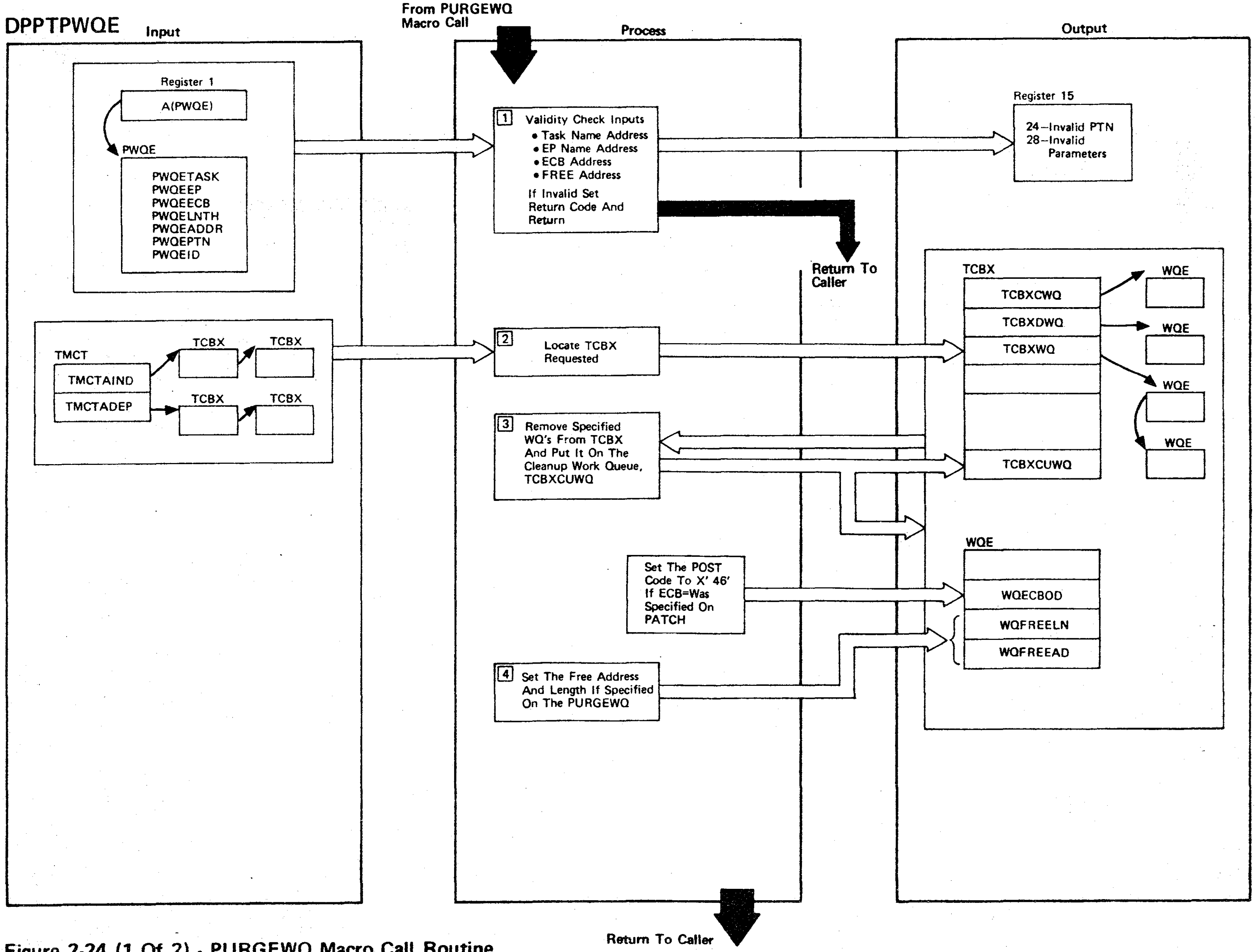


Figure 2-24 (1 Of 2) - PURGEWQ Macro Call Routine

Figure 2-24 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The PWQE address is passed in register 1. The task name address (PWQETASK), entry point name address (PWQEEP), ECB address (PWQEECB), FREE address (PWQEADDR), and the requested partition are validity checked to determine if the addresses are within the partition (or within either the MASTER or SLAVE partition in a two-partition environment).		DPPTPWQE
2	The TMCT independent task chain (TMCTAIND) and dependent task chain (TMCTADEP) are scanned to locate the specified TCBX.		DPPTPWQE
3	The EP name and ID are used to identify which work queue elements are to be removed and placed in the cleanup work queue. The work element may be on the current work queue chain (TCBXCWQ), the DEPATCH work queue chain (TCBXDWQ), or the active work queue chain (TCBXWQ). For queue holders the associated queue processors must be scanned for active work queues.		DPPTPWQE
4	The free address and length specified on the PURGEWQ are moved into the work queue element (WQFREEAD and WQFREELN) to be FREEMAINed when the work queue is detected.		DPPTPWQE

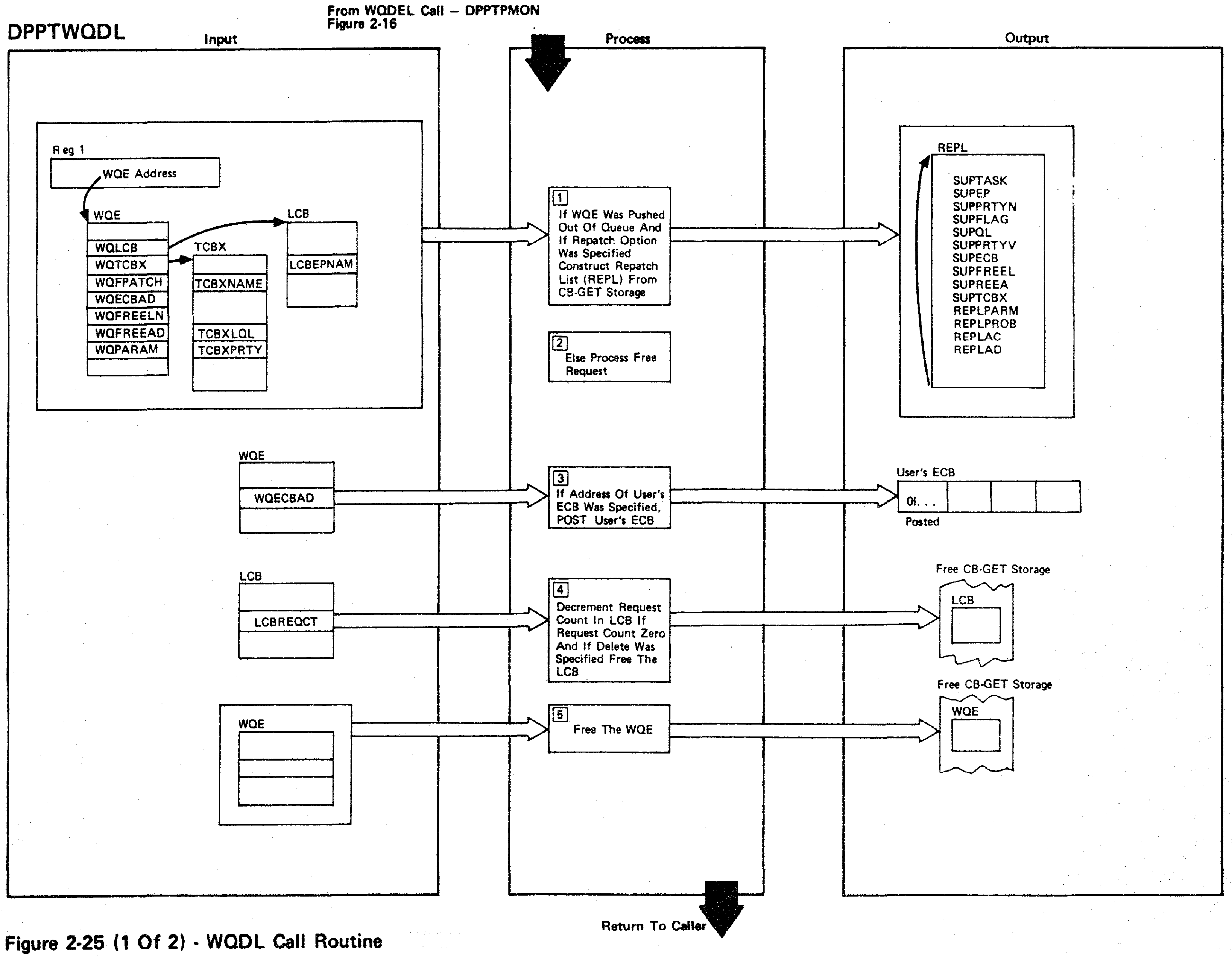


Figure 2-25 (1 Of 2) - WQDL Call Routine

LICENSED MATERIAL — PROPERTY OF IBM

Figure 2-25 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the WQE was pushed out of the queue (another PATCH with QPOS=FIRST was issued and the queue was full) and REPATCH option was specified (SUPFRPTH), a repatch list is constructed from CB-GET storage, and the parameters necessary for REPATCH are copied from TCBX, WQE, and LCB into the REPL.		DPPTWQDL
2	Otherwise, if a FREE= request was specified at PATCH time, it is processed and a FREEMAIN SVC is issued to free the user's area.		DPPTWQDL
3	If an ECB address was specified, the ECB is posted with the REPL address if step 1 above was executed; otherwise the completion code is obtained from the WQE.		DPPTWQDL
4	The request count in the LCB is decremented. If DELETE was specified and the module is reentrant, the use count in the corresponding LCB on the TMCT - LCB chain is also decremented. If it goes to zero, flags LCBFDEL and TMCTLCBD are set and DPPTSMON is posted. If DELETE was specified and the request count in the LCB is zero, the LCB is dechained and freed.		DPPTWQDL
5	The WQE must be dechained already at entry to the WQDL routine and it is freed before the routine returns to the caller.		DPPTWQDL

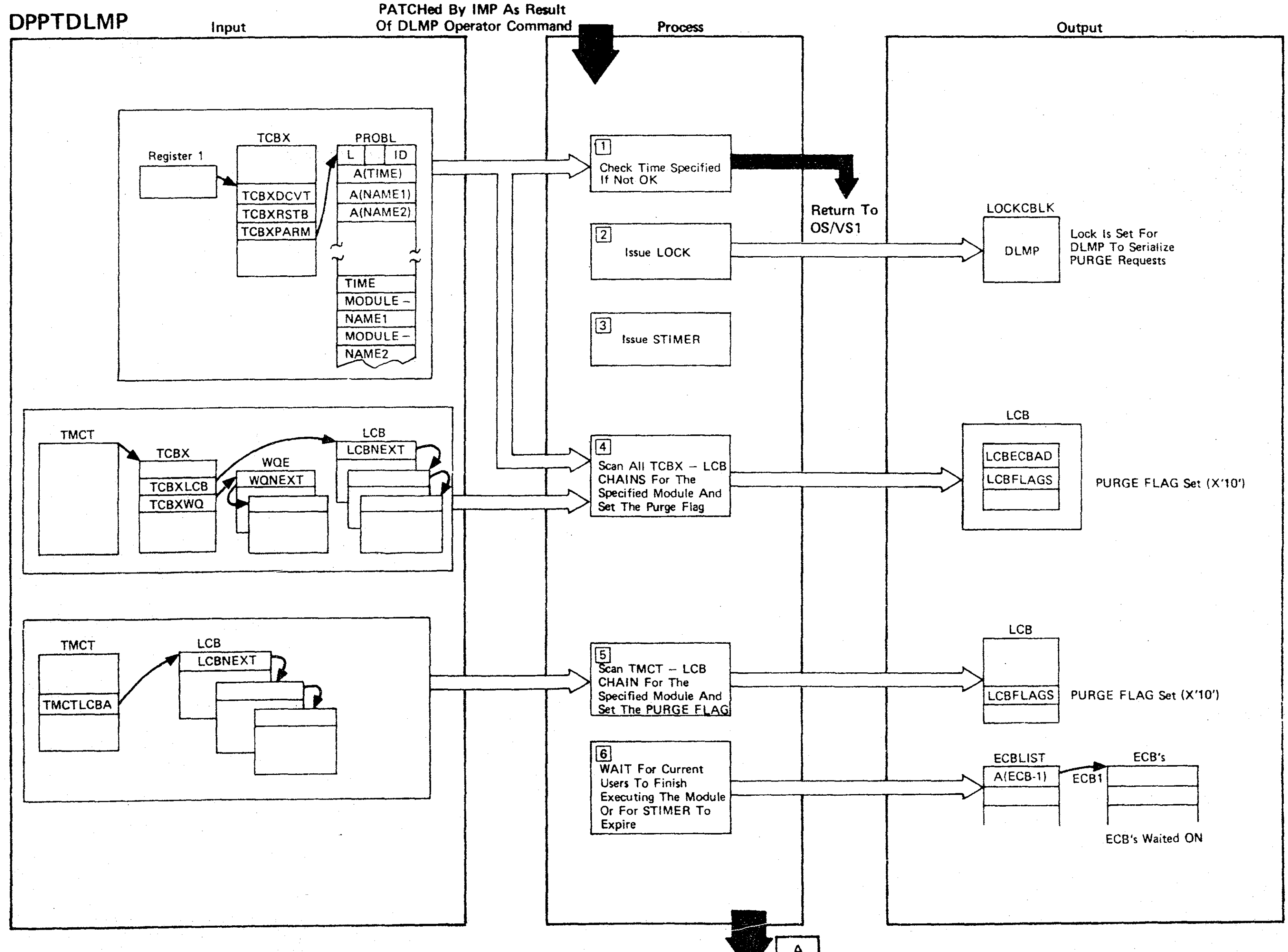


Figure 2-26 (1 Of 4) - Dynamic Load Module Purge

Figure 2-26 (2 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
	Dynamic Load Module Purge is entered as a result of a DLMP operator command through the input message processing interface.		
1	A check is made if the specified time value exceeds the maximum allowed (20 minutes); and if yes, message DPP019 is issued.	DPP019I	DPPTDLMP
2	A LOCK is issued to serialize Load Module Purge requests and message DPP020 is issued.	DPP020I	DPPTDLMP
3	A STIMER macro is issued with the specified time or a default of 2 seconds, if not specified.		DPPTDLMP
4	All TCBXs on both the independent and the dependent task chain are scanned for LCBs which reference the module names received in the purge request. If a match is found, the purge flag is set in the LCB, and if the LCB is referred to by the current WQE, an ECB is built and its address stored into the LCB.		DPTDLMP1
5	The TMCT-LCB chain is scanned for the module names received in the purge request. If a match is found, the purge flag is set in the LCB.		DPPTDLMP
6	The program waits on an ECB list for all current users of one of the modules to complete (DPPTPMON will POST the ECB) or for the STIMER issued in step 3 to expire.		DPPTDLMP

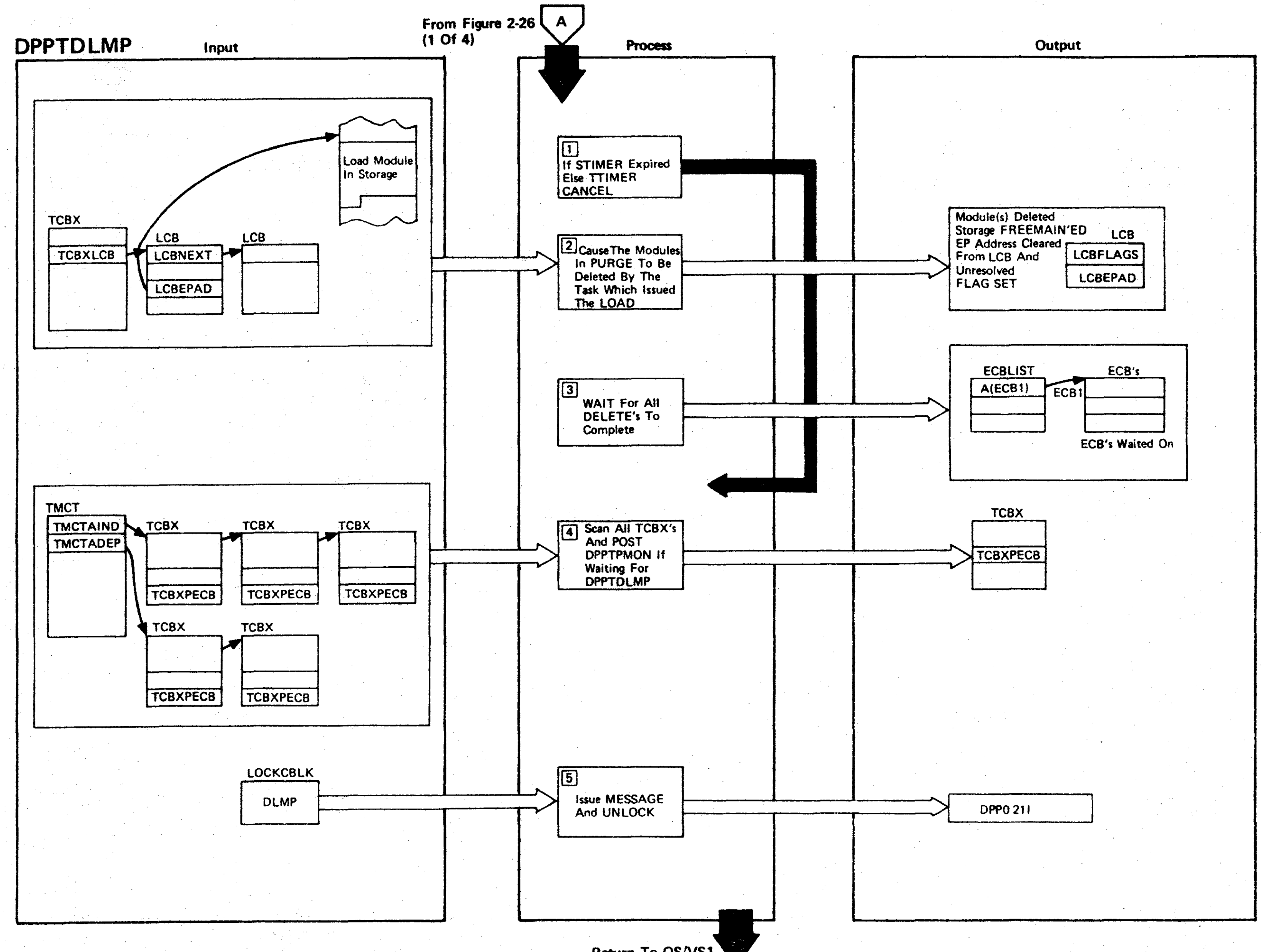
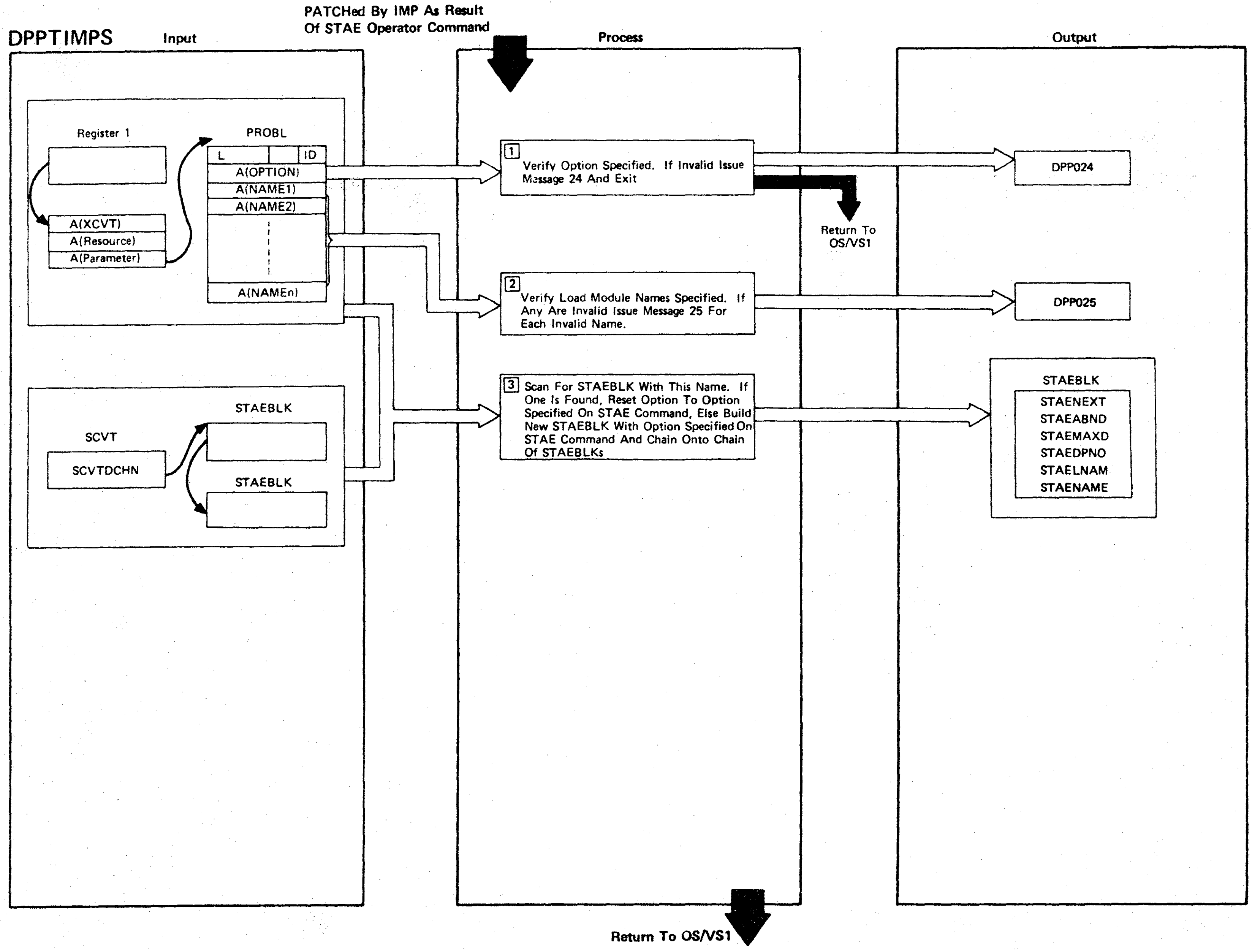

DPPTDLMP
Input
From Figure 2-26 (1 Of 4)
A
Process
Output
Load Module In Storage
TCBX
TCBXLCB
LCB
LCBNEXT
LCBEPAD
LCB
1 If STIMER Expired Else TTIMER CANCEL
2 Cause The Modules In PURGE To Be Deleted By The Task Which Issued The LOAD
Module(s) Deleted Storage FREEMAIN'ED EP Address Cleared From LCB And Unresolved FLAG SET
LCB
LCBFLAGS
LCBEPAD
3 WAIT For All DELETE's To Complete
ECBLIST
A(ECB1)
ECB1
ECB's
ECB's Waited On
TMCT
TMCTAIND
TMCTADEP
TCBX
TCBXPECB
TCBX
TCBXPECB
TCBX
TCBXPECB
TCBX
TCBXPECB
TCBX
TCBXPECB
4 Scan All TCBX's And POST DPPTPMON If Waiting For DPPTDLMP
TCBX
TCBXPECB
LOCKCBLK
DLMP
5 Issue MESSAGE And UNLOCK
DPP0 21I
Return To OS/VS1

Figure 2-26 (3 Of 4) - Dynamic Load Module Purge

Figure 2-26 (4 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the STIMER has expired issue an error message and give control to step 4 below; otherwise TTIMER CANCEL is issued.	DPP022I	DPPTDLMP
2	The modules to be purged must be deleted by the same task that issued the LOAD. The program scans all TCBX-LCBs for both the purge flag set and loaded by DPPTPMON. For each task with this condition an IRB and ECB are built and the asynchronous delete routine DPTDLMP5 issues the DELETE, clears the purge flag, and posts the ECB. Also the TMCT-LCB chain is scanned, and modules with the purge flag set are also flagged for delete, and DPPTSMON is posted to process the DELETE.		DPTDLMP2
3	DPPTDLMP waits on an ECB list for all scheduled DELETE operations to complete. A message is issued to indicate successful execution.	DPP023I	DPPTDLMP
4	Both the independent and the dependent task chain are then scanned for any DPPTPMON waiting on TCBXPECB. If waiting, TCBXPECB is posted so that DPPTPMON will resume execution.		DPTDLMP3
5	Messages are issued and UNLOCK is done, then the program returns to the caller.	DPP021I	DPPTDLMP



1 Verify Option Specified. If Invalid Issue Message 24 And Exit

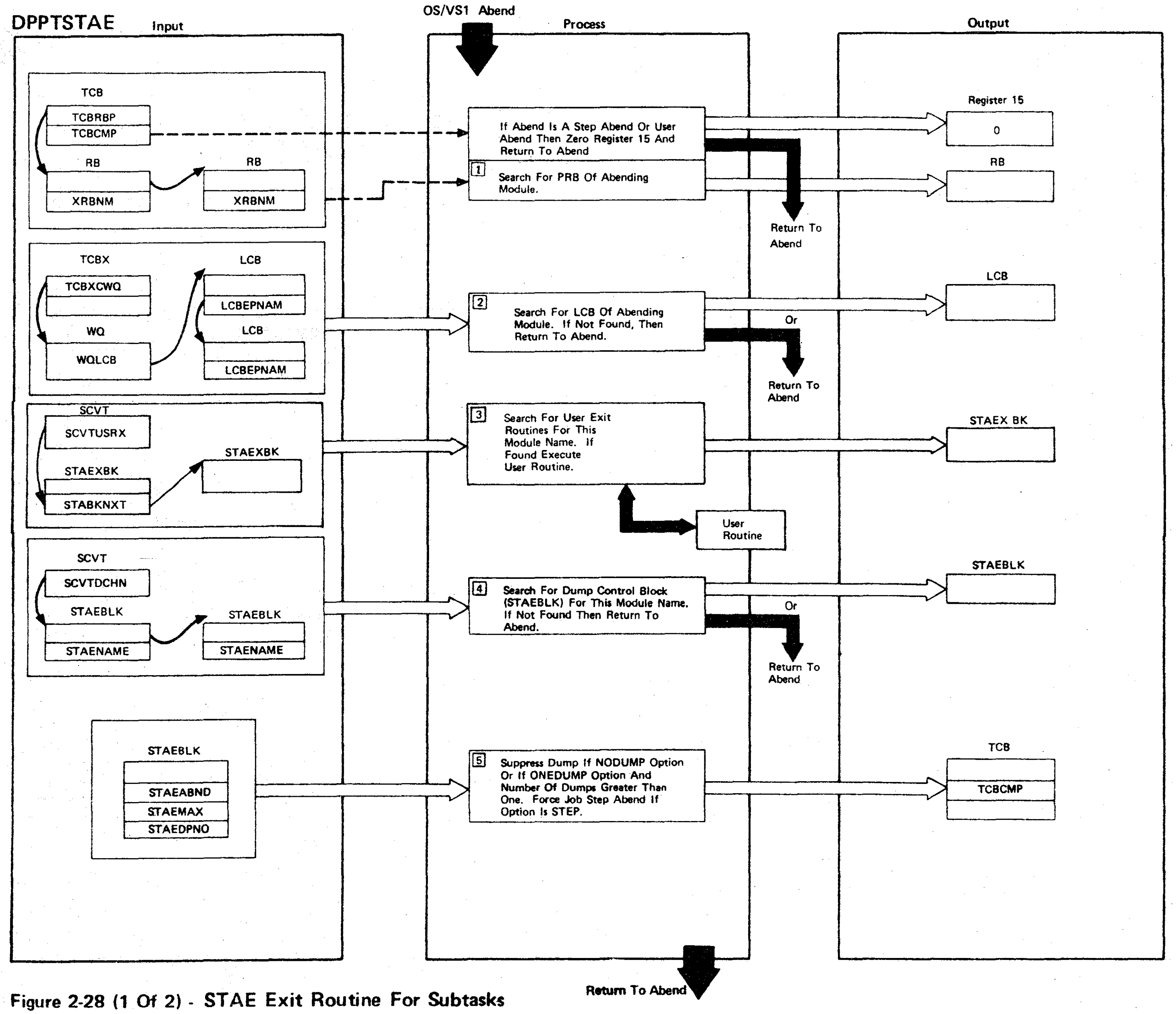
2 Verify Load Module Names Specified. If Any Are Invalid Issue Message 25 For Each Invalid Name.

3 Scan For STAEBLK With This Name. If One Is Found, Reset Option To Option Specified On STAE Command, Else Build New STAEBLK With Option Specified On STAE Command And Chain Onto Chain Of STAEBLKs

Figure 2-27 (1 of 2) STAE Command Processor - DPPTIMPS

Figure 2-27 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
	The STAE command processor is entered as a result of a STAE operator command through the Input Message Processor (IMP) interface.		
1	The valid options are DUMP, NODUMP, ONEDUMP, STEP, or OPTION.	DPP024I	DPPTIMPS
2	The load module name must be alphameric or one of the special characters $, #, or @. The first character must not be numeric.	DPP025I	DPPTIMPS
3	The STAEBLKs are chained in collating sequence.		DPPTIMPS



If Abend Is A Step Abend Or User Abend Then Zero Register 15 And Return To Abend

1 Search For PRB Of Abending Module.

2 Search For LCB Of Abending Module. If Not Found, Then Return To Abend.

3 Search For User Exit Routines For This Module Name. If Found Execute User Routine.

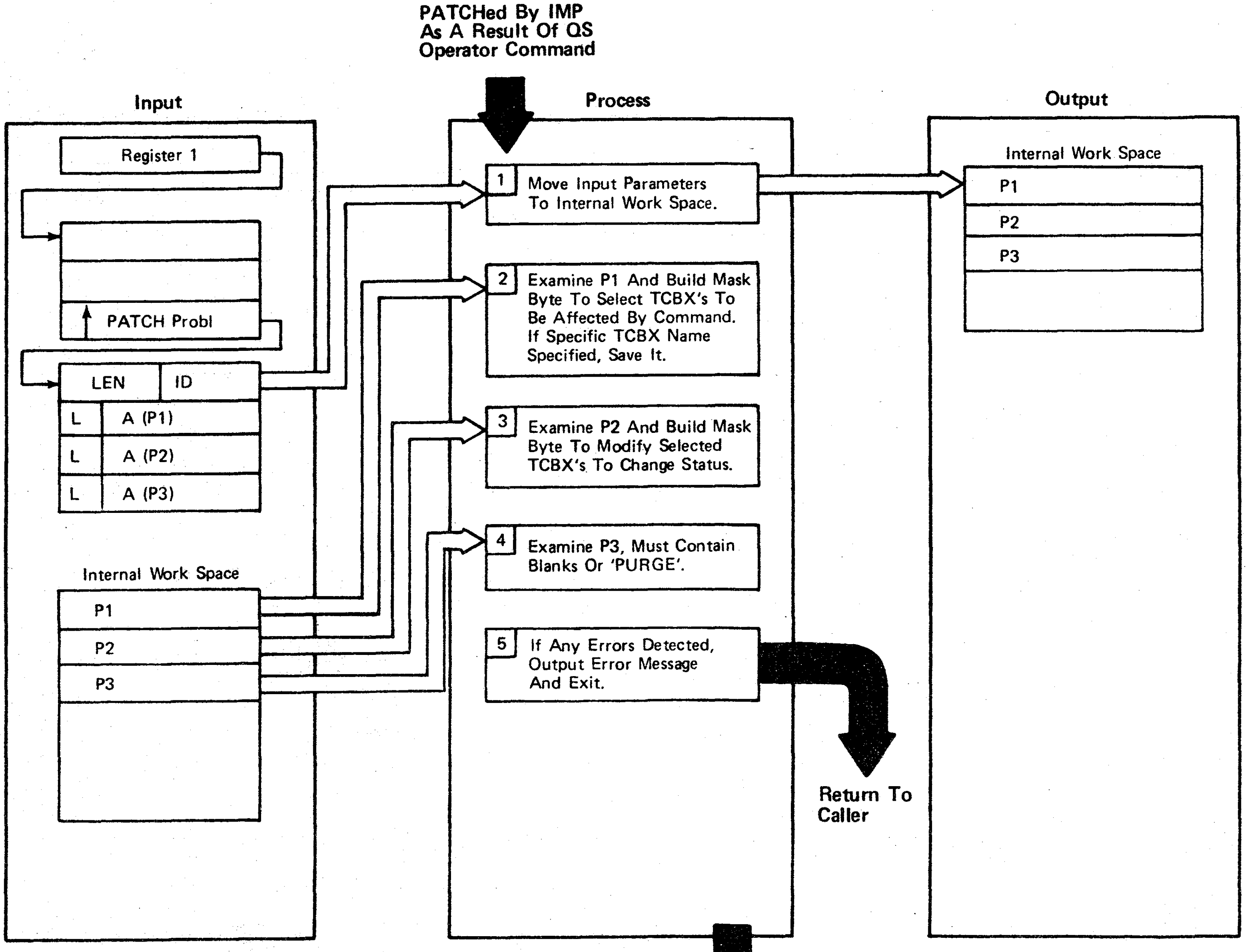
4 Search For Dump Control Block (STAEBLK) For This Module Name. If Not Found Then Return To Abend.

5 Suppress Dump If NODUMP Option Or If ONEDUMP Option And Number Of Dumps Greater Than One. Force Job Step Abend If Option Is STEP.

Figure 2-28 (1 Of 2) - STAE Exit Routine For Subtasks

Figure 2-28 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The TCB request block chain (RB) is scanned to find the first PRB whose load module name is not DPPTPMON. This is to identify any routine that has been LINKed or SYNCHed to.		DPPTSTAE
2	The work queue (WQE) load control block chain (LCB) is used to locate the entry point name of the module given control from DPPTPMON. If it is not a Special Real Time Operating System task, control is returned to ABEND processing.		DPPTSTAE
3	Using the PRB module nane (if found in step 1) or the LCB entry point nane from step 2 as the name of the ABEND module, the user STAE exit control block (STAESBR) chain is scanned to determine if a user exit routine was specified for that module. If so, the user exit routine, register 15 will contain zero if normal SRTOS STAE processing is to continue zero, if normal SRTOS is a plus for value,if OS retry is requested and a negative four value to by pass normal SRTOS STAE processing and OS retry.		
4	Using the PRB load module name (if found in step 1) or the LCB entry point name from step 2 as the name of the ABENDing module, the STAE control block (STAEBLK) chain is scanned to determine if any special processing has been requested for that module. If not, control is returned to ABEND processing.		DPPTSTAE
5	The option flags in the STAE control block (STAEABND) is used to determine the processing requested on a previous STAE command.		DPPTSTAE



1 Move Input Parameters To Internal Work Space.

2 Examine P1 And Build Mask Byte To Select TCBX's To Be Affected By Command. If Specific TCBX Name Specified, Save It.

3 Examine P2 And Build Mask Byte To Modify Selected TCBX's To Change Status.

4 Examine P3, Must Contain Blanks Or 'PURGE'.

5 If Any Errors Detected, Output Error Message And Exit.

Figure 2-28.1 (1 Of 4) - DPPTQIMP

A 2-28.1 (3 Of 4)

Figure 2-28.1 (2 Of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Standard save entry conventions are observed and space is GETMAINed for internal work space.		DPPTQIMP
2	P1 can contain any of the following: QPnn - one specific queue processor to be affected ALLQP - all queue processors to be affected ALLQH - all queue holders to be affected ALL - all queue processors, queue holders and independent tasks to be affected name - one specific task or queue holder to be affected		DPPTQIMP
3	P2 can contain any of the following: SEQ - set selected TCBX(s) to sequential state NONSEQ - set selected TCBX(s) to non sequential state HOLD - do not allow work to be started from work queue of this TCBX REL - release hold state NOPATCH - do not accept PATCHes to selected TCBXs PATCH - accept PATCHes to selected TCBXs STATUS - report status of above conditions without change XREF - report status as above plus connections between queue holders and queue processors.		DPPTQIMP
4	P3 can be omitted or contain the characters 'PURGE'. Anything else will be an error condition.		DPPTQIMP
5	Any errors detected processing P1, P2, or P3 will cause the remaining processing to be bypassed. The parameter that is in error is inserted into the message.	DPP864I	DPPTQIMP

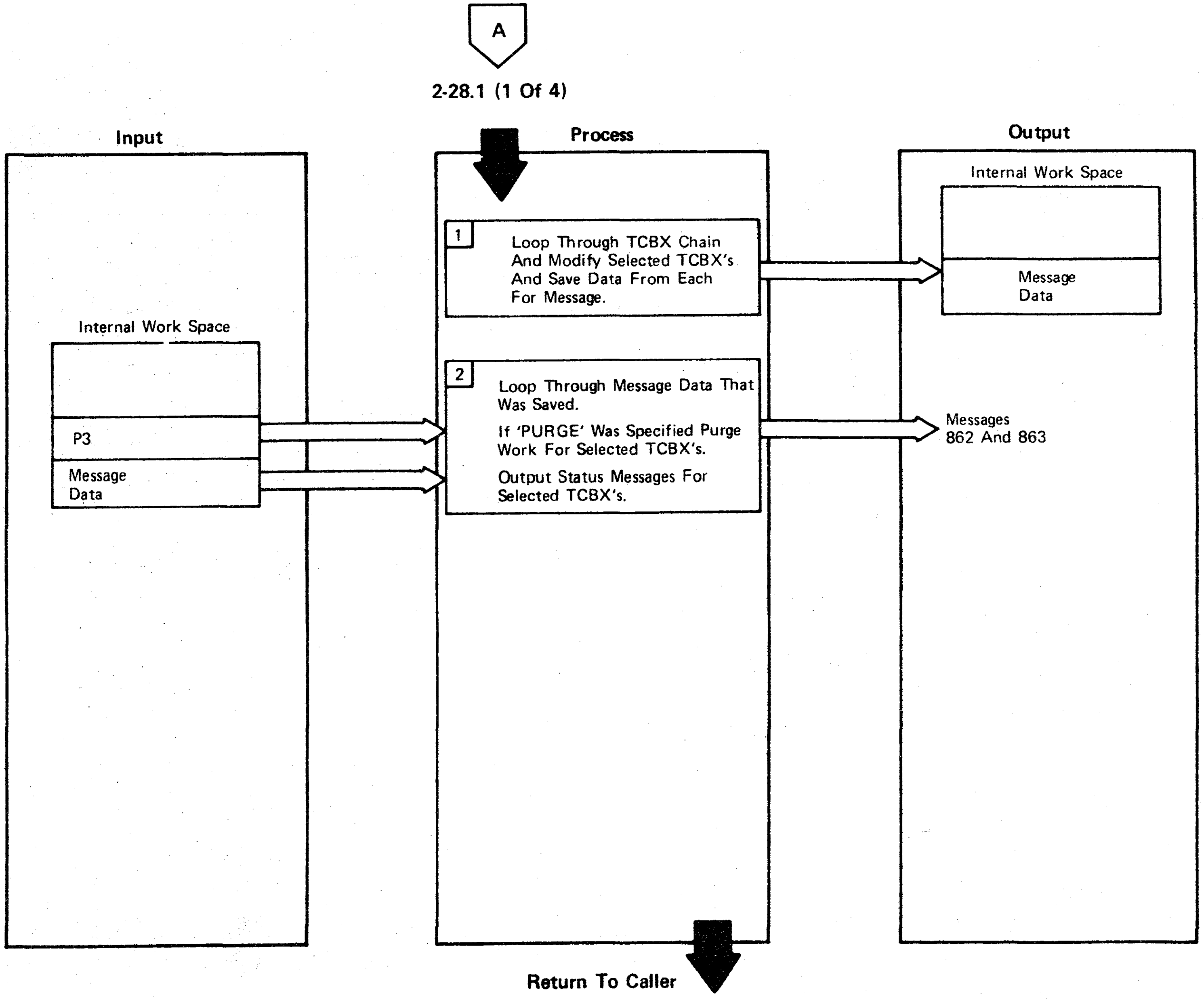

A
2-28.1 (1 Of 4)
Input
Process
Output
Internal Work Space
P3
Message Data
1
Loop Through TCBX Chain And Modify Selected TCBX's And Save Data From Each For Message.
2
Loop Through Message Data That Was Saved.
If 'PURGE' Was Specified Purge Work For Selected TCBX's.
Output Status Messages For Selected TCBX's.
Internal Work Space
Message Data
Messages 862 And 863
Return To Caller

Figure 2-28.1 (3 Of 4) - DPPTQIMP

Figure 2-28.1 (4 Of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	All TCBXs are examined to determine if they are to be affected, based on the mask byte and/or name. Those selected are modified if requested and a message data block in the internal work space built for each selected TCBX.		DPPXQIMP
2	The data collected in the message data block(s) is formatted into message DPP862I. If XREF was specified, message DPP863I is output one or more times for each selected queue holder and queue processor. It contains the names of the TCBX(s) that are connected to the selected TCBX.	DPP862I DPP863I	DPPTQIMP

Time Management

The Special Real Time Operating System time management services fall into two major categories. First, the Special Real Time Operating System time and date are maintained independently of the OS/VS1 time and date. Second, the capability of issuing PATCHes on a cyclic-time interval is provided through the PTIME macro call. This is accomplished by two subtasks created during initialization by DPPITIMI and the PTIME SVC, DPPCTSVC. The time update routine, DPPCTIME, is responsible for updating the time and date in the Special Real Time Operating System data base array, DPPCTIMA, and for posting the PTIM monitor routine, DPPCPTIM, whenever one or more PATCHes are to be issued.

The user communicates with the time management routine through a PTIME macro call. This is shown in Figure 2-29.

At initialization, or at midnight, or whenever it is determined that the time maintained by the Special Real Time Operating System is not correct, a time management routine, DPPCALCF, is called to calculate a new correction factor to be added to the time-of-day clock value to obtain the corrected time. Another routine, DPPCUPCF, is called to update the correction factor.

Serial use of the array, and this PTQE chain by the time management routines is via the use of LOCK requests specifying the resource name 'TIME'.

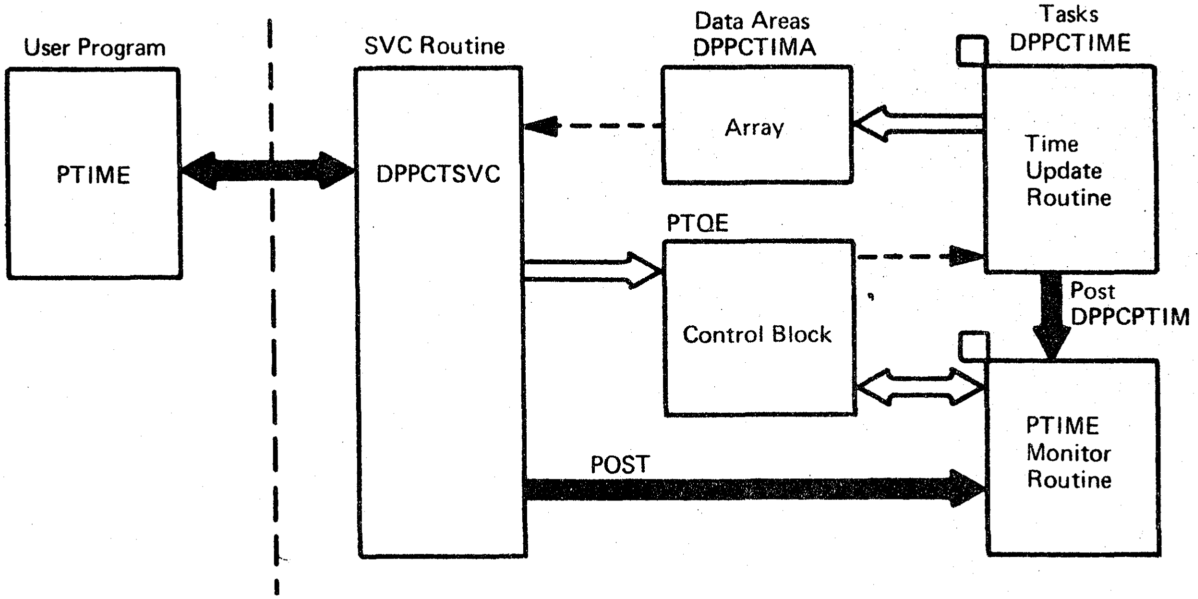


Figure 2-29. Time Management-User Program Relationship

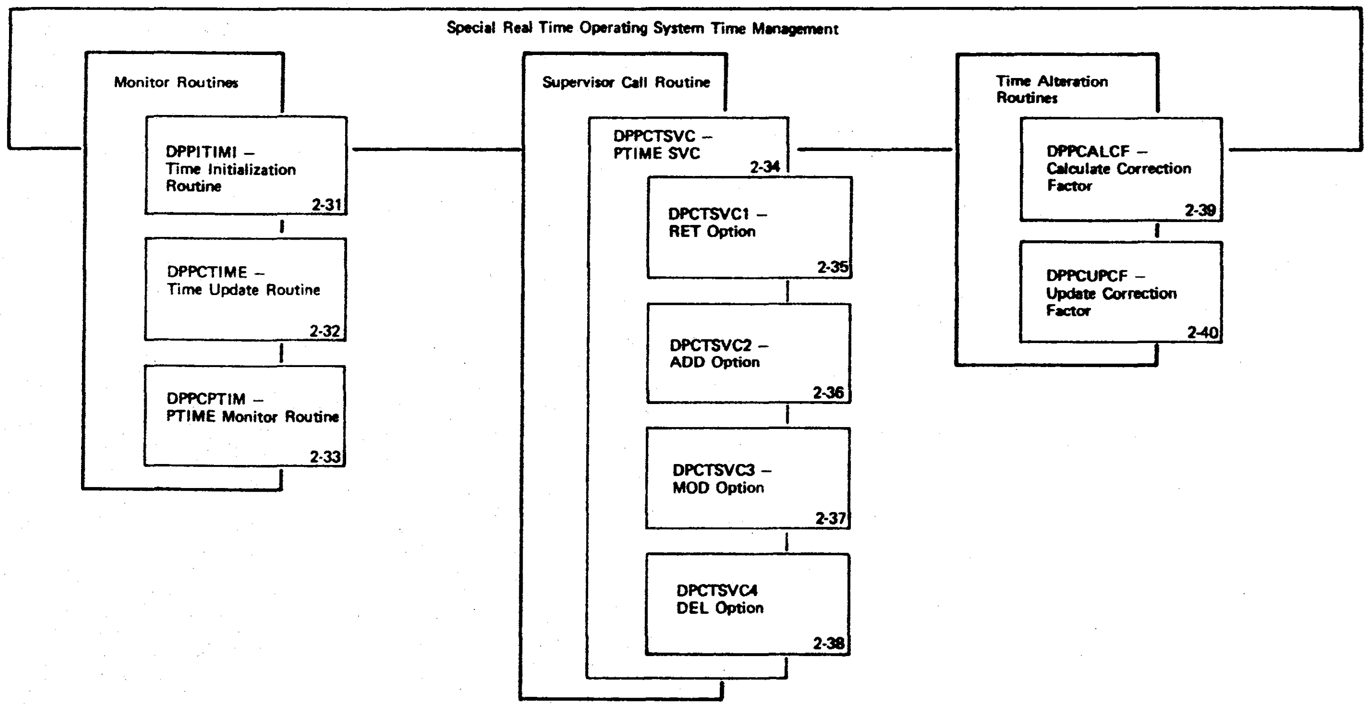

Special Real Time Operating System Time Management
Monitor Routines
DPPITIMI – Time Initialization Routine 2-31
DPPCTIME – Time Update Routine 2-32
DPPCPTIM – PTIME Monitor Routine 2-33
Supervisor Call Routine
DPPCTSVC – PTIME SVC 2-34
DPCTSVC1 – RET Option 2-35
DPCTSVC2 – ADD Option 2-36
DPCTSVC3 – MOD Option 2-37
DPCTSVC4 DEL Option 2-38
Time Alteration Routines
DPPCALCF – Calculate Correction Factor 2-39
DPPCUPCF – Update Correction Factor 2-40

Figure 2-30 (1 of 2) Special Real Time Operating System Time Management Overview

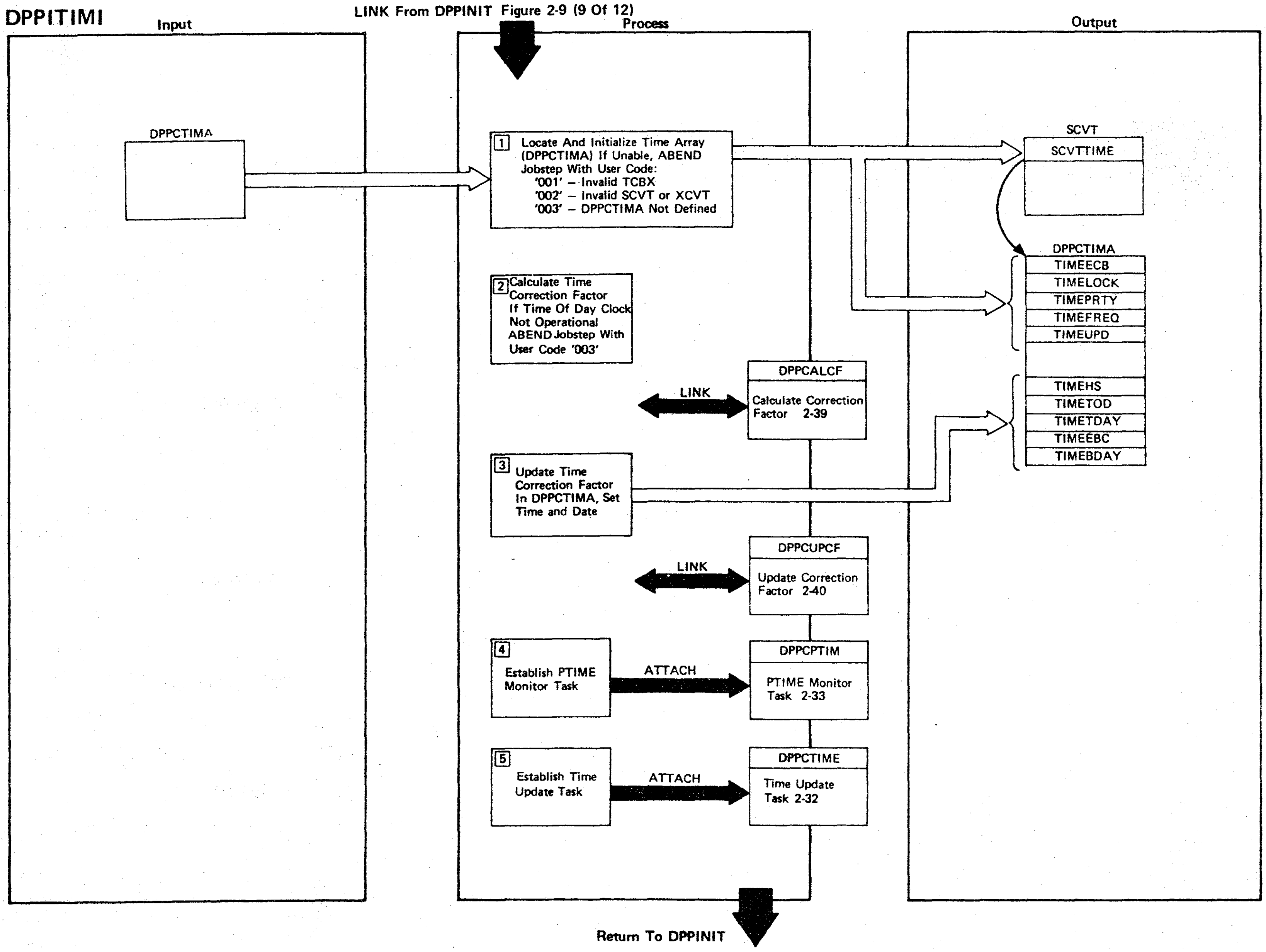

DPPITIMI
Input
LINK From DPPINIT Figure 2-9 (9 Of 12)
Process
Output
DPPCTIMA
1 Locate And Initialize Time Array (DPPCTIMA) If Unable, ABEND Jobstep With User Code:
'001' – Invalid TCBX
'002' – Invalid SCVT or XCVT
'003' – DPPCTIMA Not Defined
2 Calculate Time Correction Factor If Time Of Day Clock Not Operational ABEND Jobstep With User Code '003'
LINK
DPPCALCF
Calculate Correction Factor 2-39
3 Update Time Correction Factor In DPPCTIMA, Set Time and Date
LINK
DPPCUPCF
Update Correction Factor 2-40
4 Establish PTIME Monitor Task
ATTACH
DPPCPTIM
PTIME Monitor Task 2-33
5 Establish Time Update Task
ATTACH
DPPCTIME
Time Update Task 2-32
SCVT
SCVTTIME
DPPCTIMA
TIMEECB
TIMELOCK
TIMEPRTY
TIMEFREQ
TIMEUPD
TIMEHS
TIMETOD
TIMETDAY
TIMEEBC
TIMEBDAY
Return To DPPINIT

Figure 2-31 (1 of 2) Time Management Initialization - DPPITIMI

Figure 2-31 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	A GETARRAY macro call is used to obtain the address of the time array, DPPCTIMA. This address is stored into the SCVT.	USER 1 USER 2 USER 4	DPPITIMI
2	Module DPPCALCF is entered via a LINK SVC to calculate a time correction factor. The condition code is tested after a "store clock" instruction to determine if the TOD clock is operational.	USER 3	DPPITIMI
3	Module DPPCUPCF is entered via a LINK SVC to update the time correction factor and set the current time in the time array.		DPPITIMI
4	Module DPPCPTIM is attached to create the PTIME monitor task.		DPPITIMI
5	Module DPPCTIME is attached to create the time update task.		DPPITIMI

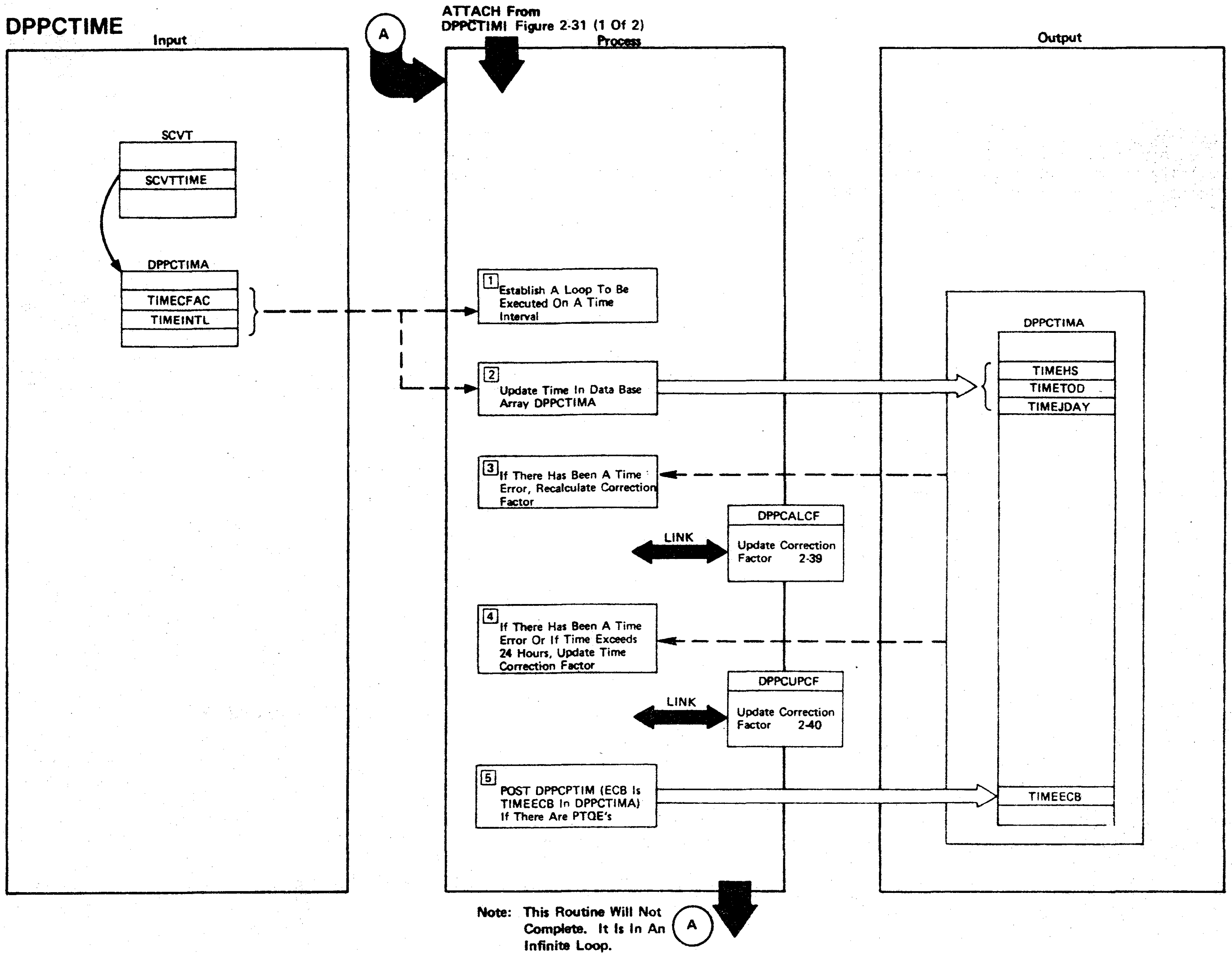
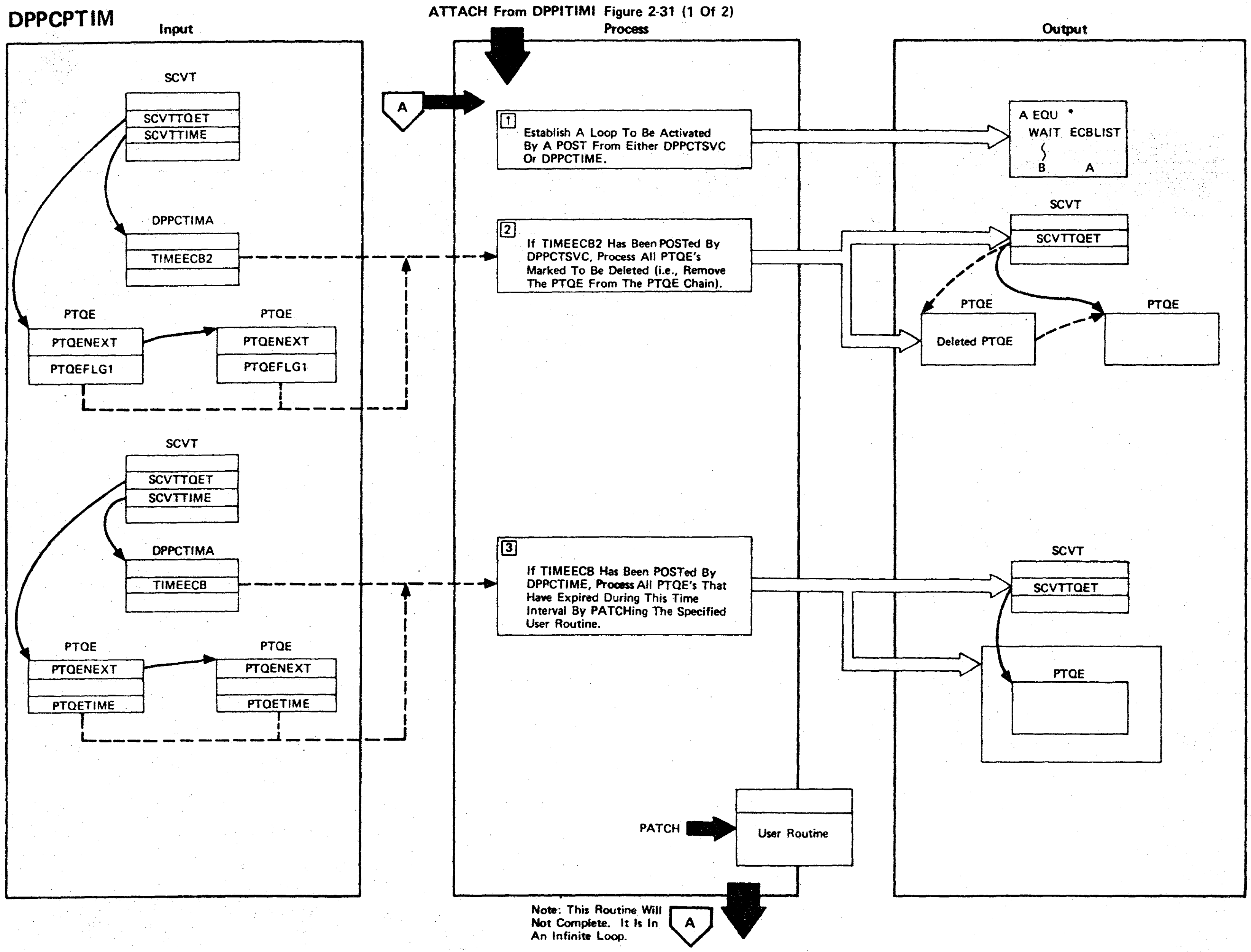

DPPCTIME
Input
ATTACH From
DPPCTIMI Figure 2-31 (1 Of 2)
Process
Output
A
SCVT
SCVTTIME
DPPCTIMA
TIMECFAC
TIMEINTL
1 Establish A Loop To Be Executed On A Time Interval
2 Update Time In Data Base Array DPPCTIMA
3 If There Has Been A Time Error, Recalculate Correction Factor
DPPCALCF
LINK
Update Correction Factor 2-39
4 If There Has Been A Time Error Or If Time Exceeds 24 Hours, Update Time Correction Factor
DPPCUPCF
LINK
Update Correction Factor 2-40
5 POST DPPCPTIM (ECB Is TIMEECB In DPPCTIMA) If There Are PTQE's
DPPCTIMA
TIMEHS
TIMETOD
TIMEJDAY
TIMEECB
Note: This Routine Will Not Complete. It Is In An Infinite Loop.
A

Figure 2-32 (1 Of 2) - Time Management Time Update Routine

Figure 2-32 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	A STIMER WAIT is issued specifying the SYSGENed time interval. After all processing has been completed, DPPCTIME branches back to the top of the program and reissues the STIMER. The time interval is contained in the time array, DPPCTIMA, which was defined during SYSGEN.		DPPCTIME
2	The OS/VS1 time-of-day clock value and the time correction value are used to calculate the current time of day.		DPPCTIME
3	If the Special Real Time Operating System time is less than or greater than the expected time by a predefined tolerance value, DPPCTIME links to DPPCALCF to recalculate the correction factor. Message 38 is issued to inform the user of this condition.	DPP038I	DPCTIME2
4	If the Special Real Time Operating System time is greater than 24 hours, a 24-hour value is subtracted from the correction value, and DPPCTIME links to DPPCUPCF to update the Special Real Time Operating System time array DPPCTIMA with the new correction factor, time, and date. If the Special Real Time Operating System time was found to be in error in step 3, DPPCTIME LINKs to DPPCUPCF to update the Special Real Time Operating System time array with the new correction factor, time, and date.		DPCTIME1
5	The TIMEECB ECB is posted . Module DPPCPTIM WAITs on this ECB. When posted , DPPCPTIM processes all PTQEs in the time interval.		DPPCTIME

DPPCPTIM



1 Establish A Loop To Be Activated By A POST From Either DPPCTSVC Or DPPCTIME.

2 If TIMEECB2 Has Been POSTed By DPPCTSVC, Process All PTQE's Marked To Be Deleted (i.e., Remove The PTQE From The PTQE Chain).

3 If TIMEECB Has Been POSTed By DPPCTIME, Process All PTQE's That Have Expired During This Time Interval By PATCHing The Specified User Routine.

Note: This Routine Will Not Complete. It Is In An Infinite Loop.

Figure 2-33 (1 Of 2) - Time Management PTIME Monitor Routine

Figure 2-33 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	DPPCPTIM waits on an ECBLIST (TIMEECB & TIMEECB2). After all processing has been completed, DPPCPTIM branches back to the top of the program and reissues the WAIT.		DPPCPTIM
2	Bit 7 of the PTQEFLG1 is used to determine if the PTQE is to be deleted. A DPATCH is issued if the user had requested it. The PATCH ECB is posted with an X'4F' if the user had supplied an ECB. The problem parameter list (if any) is then freed. The PTQE is removed from the PTQE chain and the CBGET core is freed.		DPPCPTIM
3	All PTQEs with a time of next PATCH value (PTQETIME) less than the current Special Real Time Operating System time plus the SYSGENed time interval are processed. That is, a PATCH is issued specifying the TASK as defined in the PTIME macro. If this is the last PATCH requested or if the PATCH return code is greater than 8, a DPATCH is issued if the user had requested it. The PATCH ECB is posted with an X'4F' if the user had supplied an ECB. The problem parameter list (if any) is then freed. The PTQE is removed from the PTQE chain and the CBGET core is freed. If the PATCH return code is greater than 8, an error message is issued.	DPP06II	DPPCPTIM

DPPCTSVC

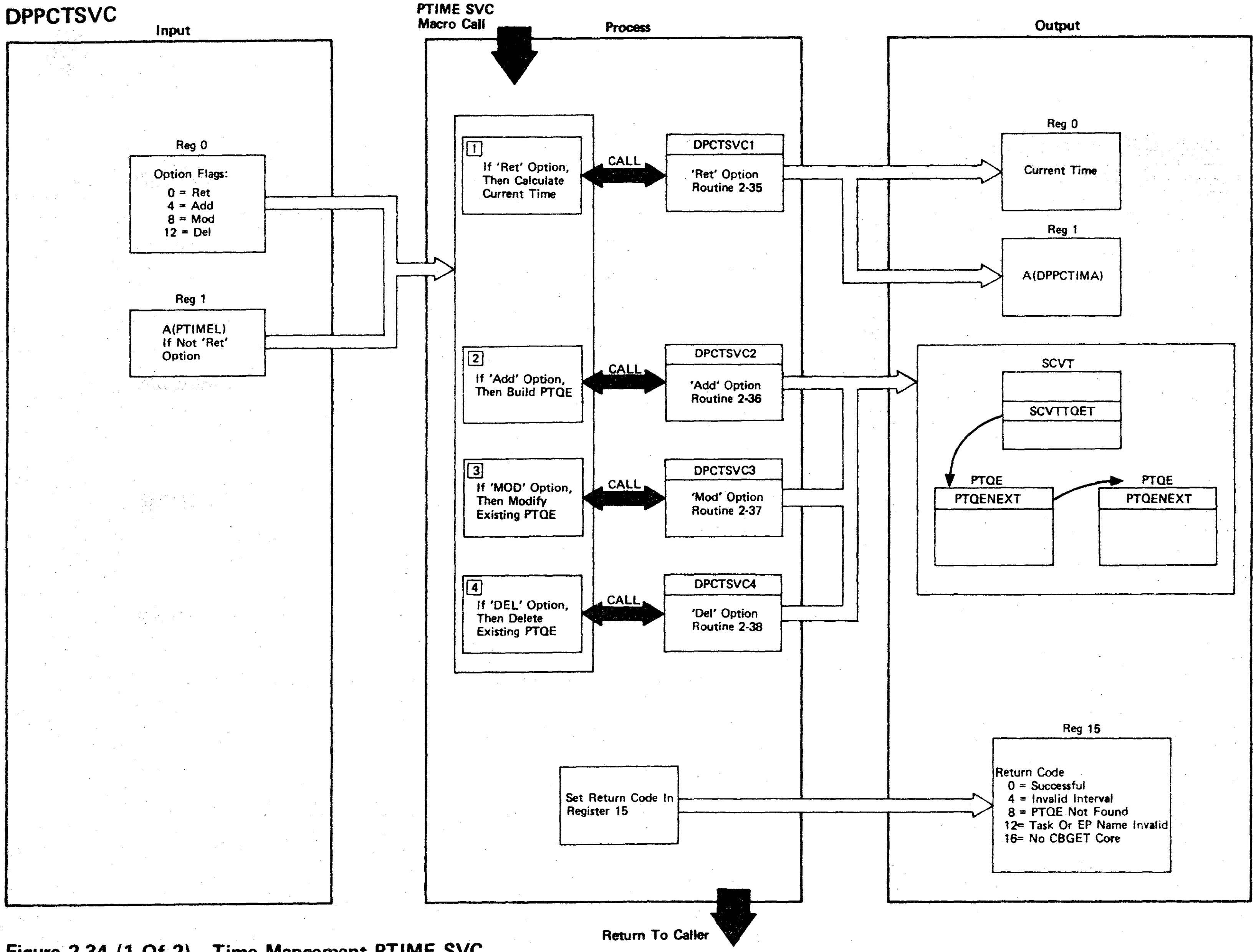


Figure 2-34 (1 Of 2) - Time Mangement PTIME SVC

Figure 2-34 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Call subroutine DPCTSVC1 to calculate current time.		DPPCTSVC
2	Call subroutine DPCTSVC2 to build a new PTQE.		DPPCTSVC
3	Call subroutine DPCTSVC3 to modify an existing PTQE.		DPPCTSVC
4	Call subroutine DPCTSVC4 to delete an existing PTQE.		DPPCTSVC

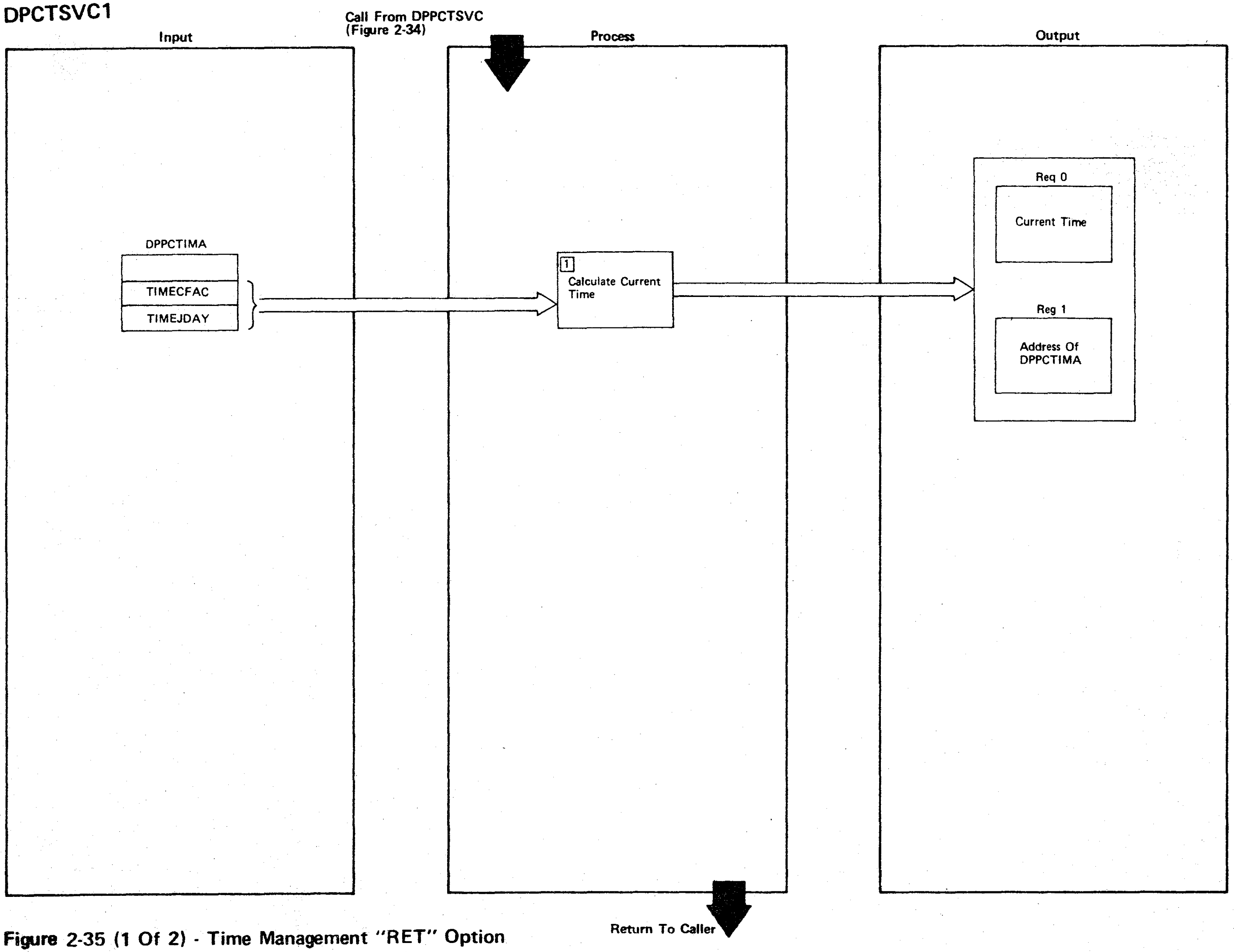


Figure 2-35 (1 Of 2) - Time Management "RET" Option

Figure 2-35 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	On entry to DPPCTSVC, general purpose register 1 contains 0 to indicate a RET PTIME option request. The Special Real Time Operation System correction factor is subtracted from the OS TOD clock value to obtain the current Special Real Time Operation System time.		DPCTSVC1

DPCTSVC2

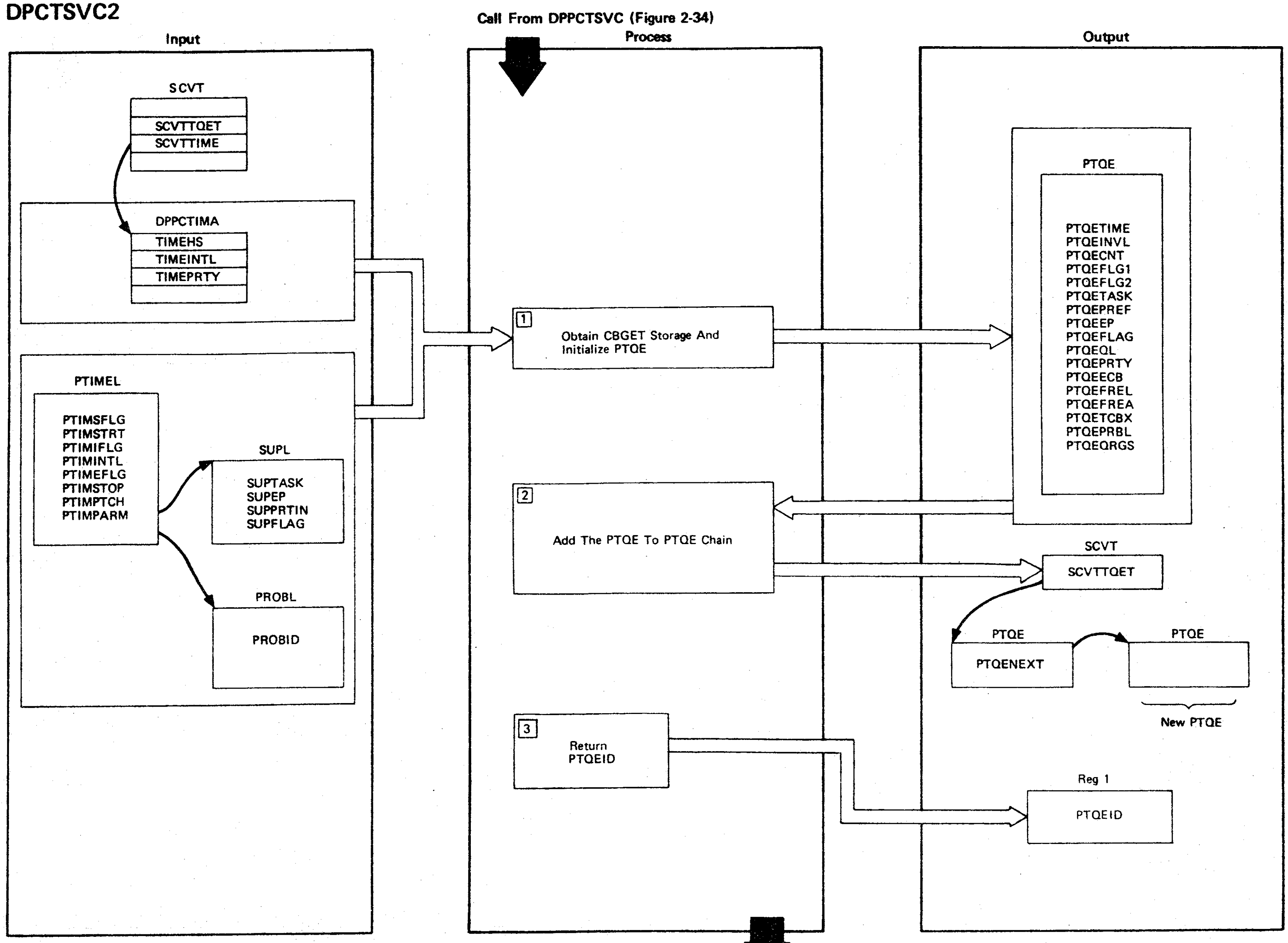


Figure 2-36 (1 Of 2) - Time Management "ADD" Option

Figure 2-36 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	On entry to DPPCTSVC, general purpose register 0 contains 4 to indicate an ADD PTIME option request, and register 1 contains the address of the PTIME input parameter list, PTIMEL. If the PROBL length is less than 8, it is saved in the PTQE. The SYSGENed time interval is the minimum acceptable value for the start, stop, or interval times. The user is informed of this condition through a return code in register 15. (See Figure 2-34.) If neither a stop time nor a count value is specified, the PTIME is assumed to be infinite.		DPCTSVC2
2	The newly created PTQE is added to a chain of PTQEs via a CHAIN macro call. This PTQE chain is ordered in ascending sequence according to the value in the PTQETIME field. If a PTQE ID was not specified, then the storage address of the PTQE is used for the PTQE ID.		DPCTSVC2
3.	The PTQE ID is returned to the caller in register 1.		

DPCTSVC3

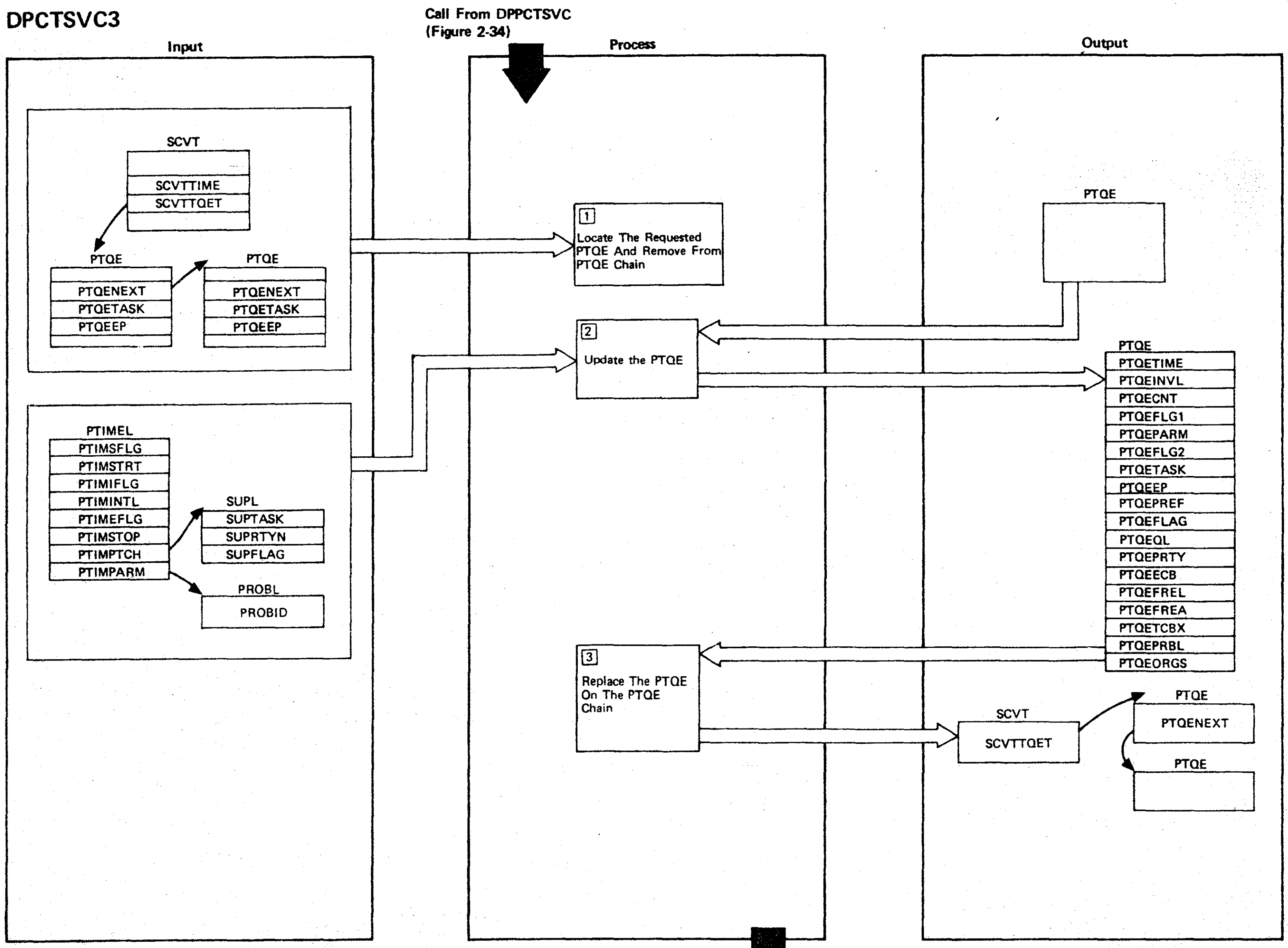


Figure 2-37 (1 Of 2) - Time Management "MOD" Option

Figure 2-37 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	On entry to DPPCTSVC, general purpose register 0 contains 8 to indicate a MOD PTIME option request, and register 1 contains the address of the PTIME input parameter list PTIMEL.		DPCTSVC3
	The PTQE chain is searched in order to locate the correct PTQE (or PTQEs). Either the task name and/or entry point must have been specified in the PTIME macro. All PTQEs containing the specified task name, and/or entry point name, and/or ID are modified. If a PTQEID is not specified. If a PTQEID is supplied then only that PTQE is modified.		DPCTSVC3
2	The PTQEs are rebuilt from this information contained in the PTIMEL.		DPCTSVC3
3	The update PTQEs are added to a chain of PTQEs via a CHAIN macro call.		

LICENSED MATERIAL — PROPERTY OF IBM

DPCTSVC4

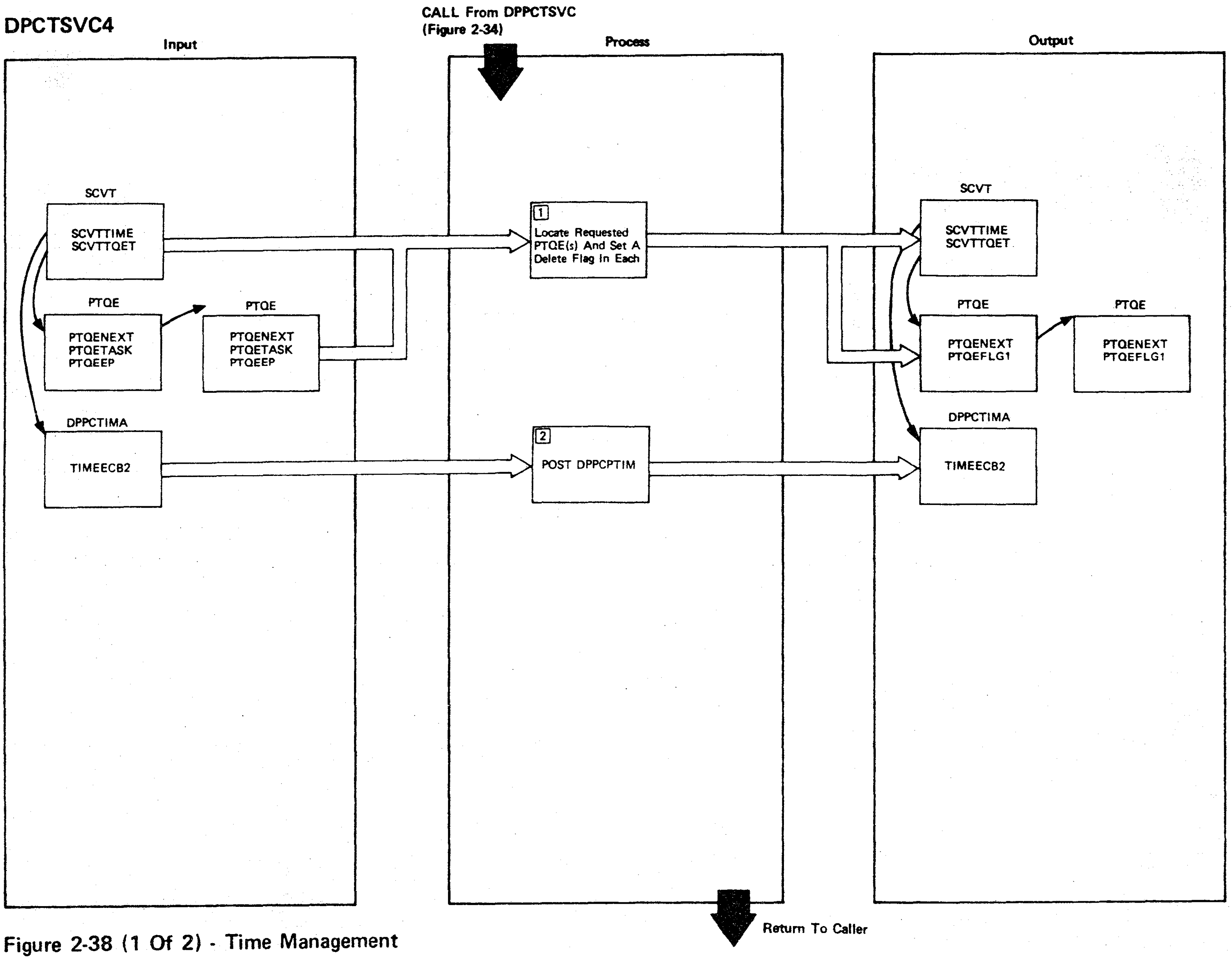


Figure 2-38 (1 Of 2) - Time Management

Figure 2-38 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	On entry to DPPCTSVC, general purpose register 0 contains 12 to indicate a DEL PTIME option request, and register 1 contains the address of the PTIME input parameter list PTIMEL. The PTQE chain is searched in order to locate the correct PTQE (or PTQEs). Either the task name and/or entry point must have been specified in the PTIME macro. All PTQEs containing the specified task name, and/or entry point name, and/or ID are modified (i.e., bit 7 of the PTQEFLG1 is turned on to indicate that this PTQE is to be deleted), if a PTQE ID is not specified. IF a PTQE ID is supplied then only that PTQE is modified.		DPCTSVC4
2	The TIMEECB ECB is posted. Module DPPCPTIM waits on this ECB. When posted, DPPCPTIM removes all PTQEs with bit 7 of the PTQEFLG1 set to one.		DPCTSVC4

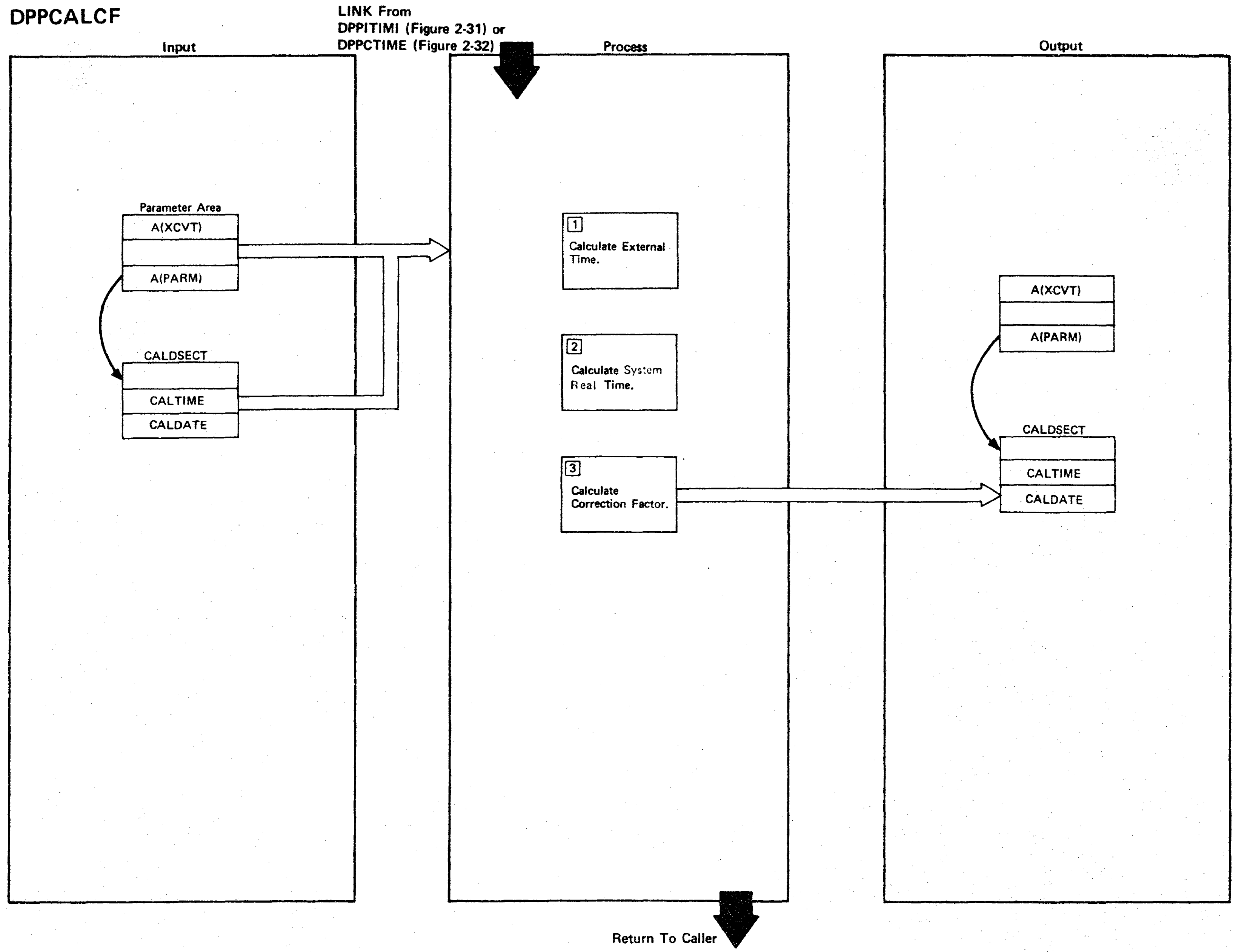


Figure 2-39 (1 Of 2) - Time Management - Calculate Correction Factor

Figure 2-39 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Register 1 contains the address of a 3-word parameter area. The time and date obtained from the external time source is stored in CALTIME and CALDATE, respectively. The time is binary in 10 millisecond units. The date is a Julian date of the form "OOYYDDDF" where YY is the last two digits of the year and DDD is the day of the year. NOTE: The default external time source is the standard OS time routine. Segment DPCALCF1 may be replaced by a user written interface program to support another time source.		DPCALCF1
2	A PTIME macro call with the RET option is used to obtain the current time.		DPPCALCF
3	The time is subtracted from the time provided by the external time source to provide an algebraic sum to be added to the current correction factor.		DPPCALCF

DPPCUPCF

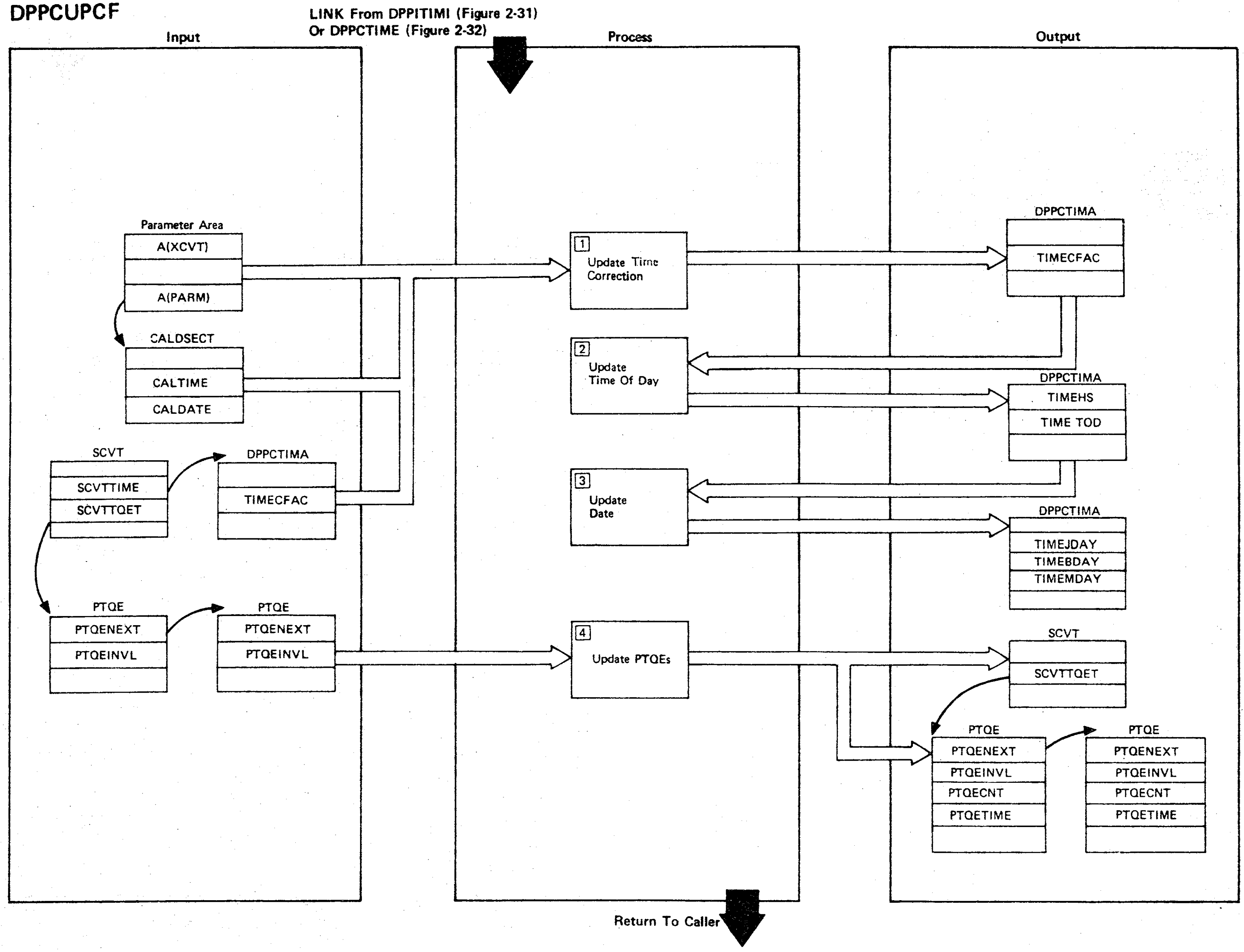


Figure 2-40 (1 Of 2) - Time Management - Update Correction Factor Routine

Figure 2-40 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Register 1 contains the address of a 3-word parameter area. The CALTIME field contains an algebraic sum to be added to the current correction factor.		DPPCUPCF DPCUPCF1
2	The current time is updated based on the new correction factor.		DPCUPCF2
3	The date stored in the CALDATE field is stored into the time array as the current date.		DPCUPCF3
4	If the time is adjusted backward, then the time and count values in the PTQEs are reset (back to the original start time if necessary). If the time is adjusted forward, then it is assumed that the intervening time intervals were skipped. NOTE: Message 39 is issued to inform the user that the time correction factor has been updated.	DPP039I	DPCUPCF4

Data Base Management

The Special Real Time Operating System data base is designed to fulfill the needs of data storage and access of a realtime operating system. The Special Real Time Operating System data base subroutines provide the user with an interface to the information contained in the data base. Through the use of these subroutines, data, may be retrieved from or replaced in the data base. In addition, sections of the data base may be copied to a direct access device to provide a historical log.

During a normal start, i.e., when the job is initially started through standard OS/VS Job Control statements with the EXEC card specifying PGM=DPPINIT, the data base initialization programs will read in the initial data for all VS resident arrays that specified "INIT=YES" on the ARRAY macro in the offline utility phase. Those VS arrays for which "INIT=YES" was not specified have VS storage space allocated, but no data is moved into the space.

During a refresh start, i.e., when the job is reinitialized from a restart data set, or during a normal start when the SYSINIT input stream does not contain a "DBREF NO" control statement, the data base initialization program will refresh all VS resident arrays that specified "REINIT=YES" and that requested logging in the offline utility phase with the last logged copy of that array. The log arrays are initialized to resume logging with the last logged copy of each loggable VS resident array.

The Data Base Initialization program, DPPIDBAS, is responsible for the initial load of the VS resident data base, building the data base control blocks, and loading the data base subroutines (DPPDBLOK (GETBLOCK/PUTBLOCK), DPPDITEM (GETITEM/PUTITEM), and DPPDARAY (GETARRAY/PUTARRAY)). These subroutines are independent with little or no communication with each other and provide the user interface with the data contained in the data base as shown in Figure 2-41.

The Data Base Logging Initialization program, DPPILOGN, is responsible for loading the logging subroutines (DPPDGETL (GETLOG), DPPDPUTL (PUTLOG), and DPPDUMPL (DUMPLOG), initiating time-driven logging (DPPDFREQ), and refreshing user-specified VS resident arrays (DPPDUPDL). The three logging subroutines are also independent of each other but use GETBLOCK, PUTBLOCK, etc. to actually retrieve the requested data as shown in Figure 2-42.

Data base is the only functional area that requires special routines used primarily for communications between partitions in a two-partition environment. Since the OS/VS1 I/O control blocks used to read and write data from the DA resident data base exist only in the MASTER partition, any data base request must be executed by a task in the MASTER partition. This is accomplished by a SLAVE partition interface routine, DPPDSUB2, which receives control in the SLAVE partition as the result of a user macro call (i.e., GETBLOCK, PUTLOG, etc.) DPPDSUB2 PATCHes a MASTER partition interface routine, DPPDBSIF, in the MASTER partition. DPPDBSIF then branches to the appropriate subroutine (i.e., DPPDBLOK, DPPDPUTL, etc.) to perform the requested service as shown in Figure 2-43.

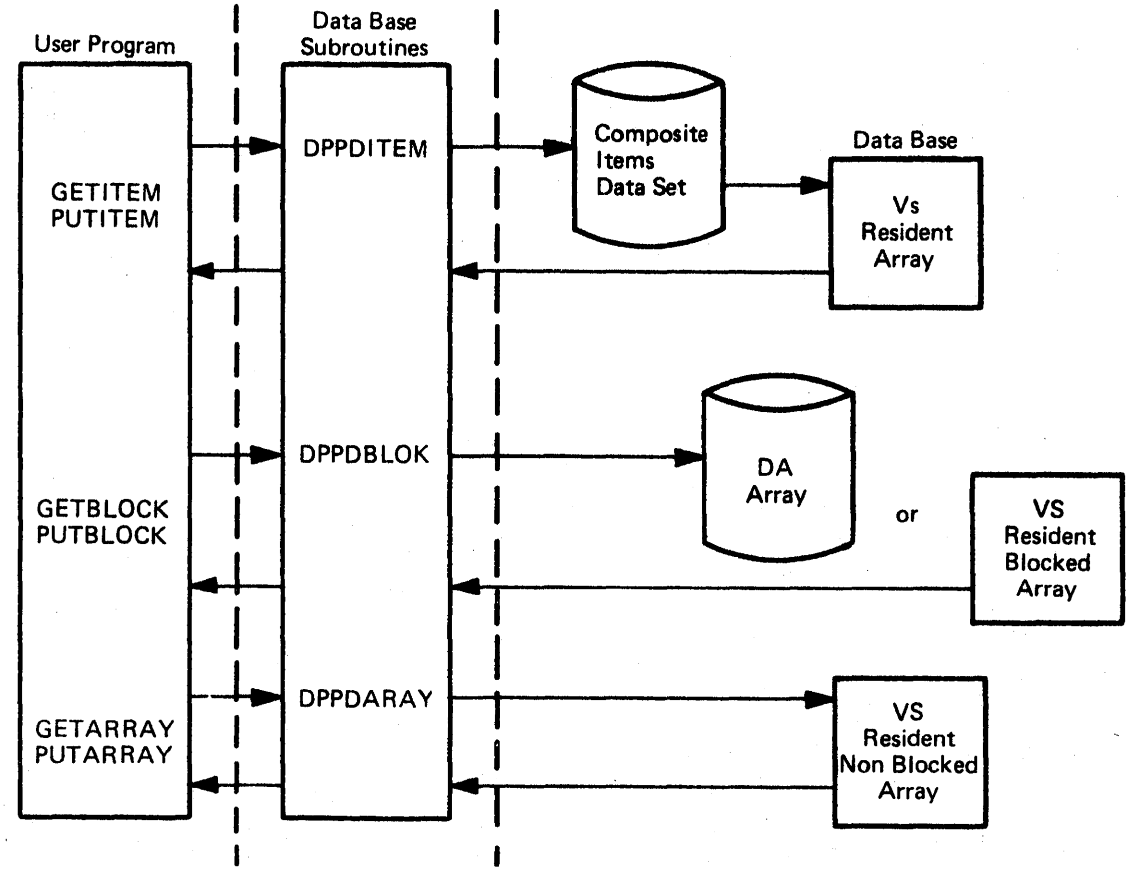


Figure 2-41. Data Base Subroutines

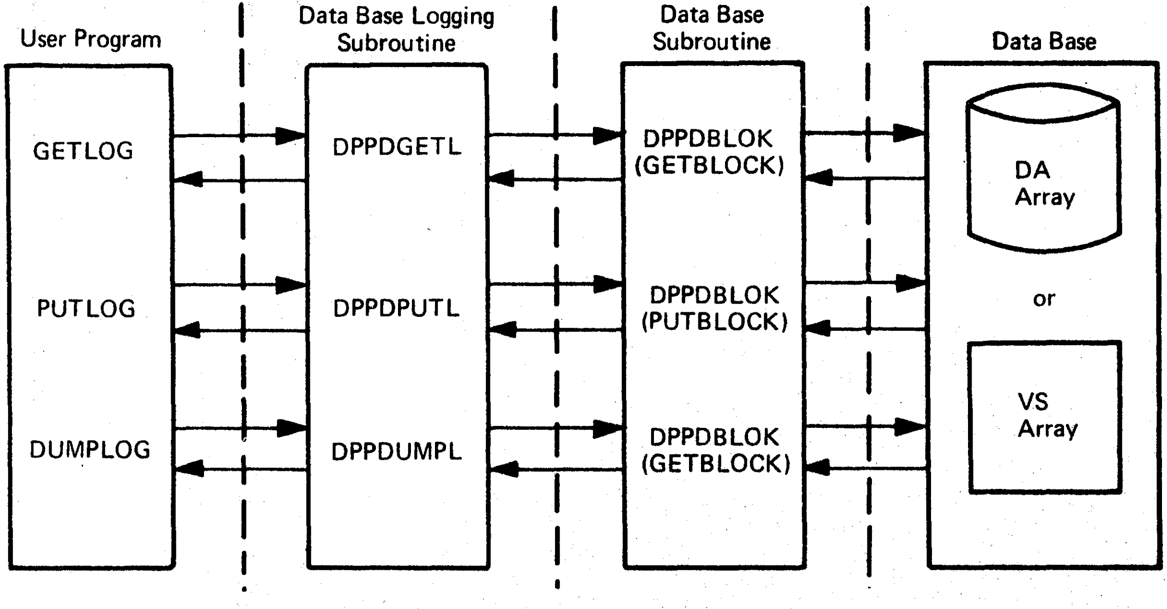


Figure 2-42. Data Base Logging Subroutines

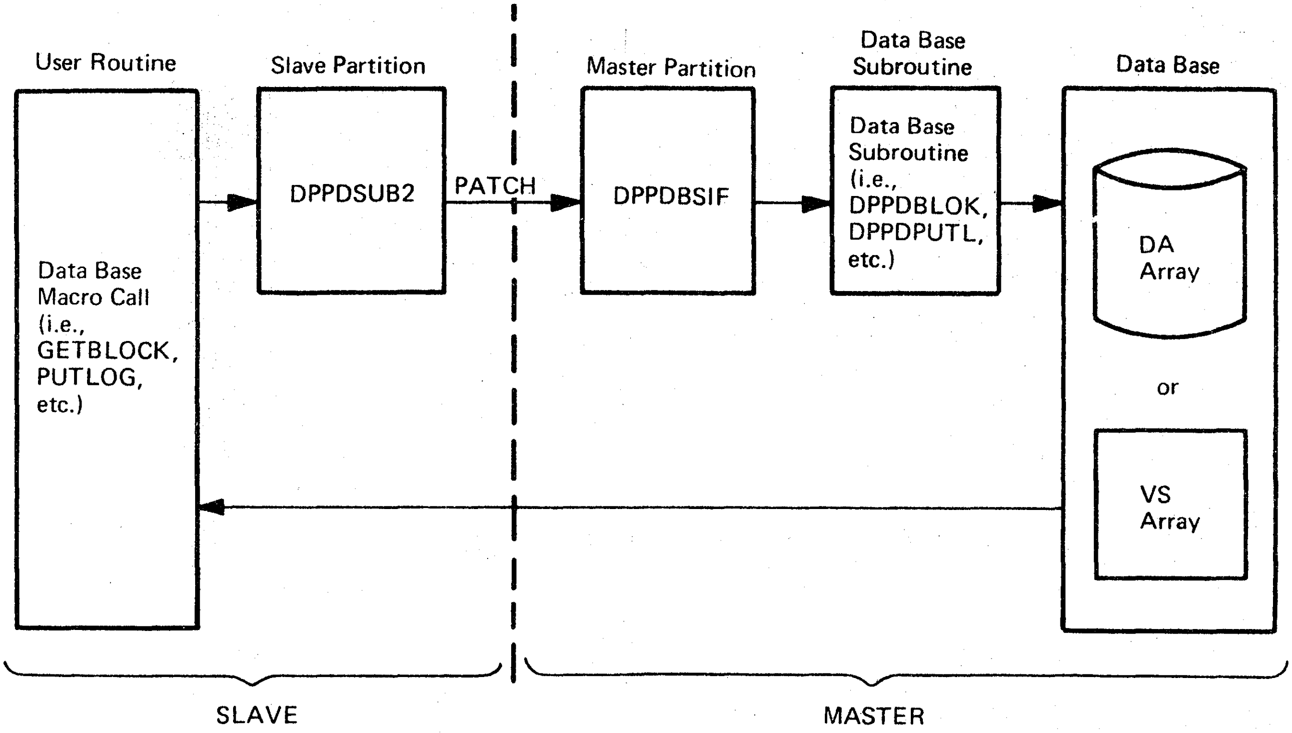


Figure 2-43. Data Base Two Partition Operation

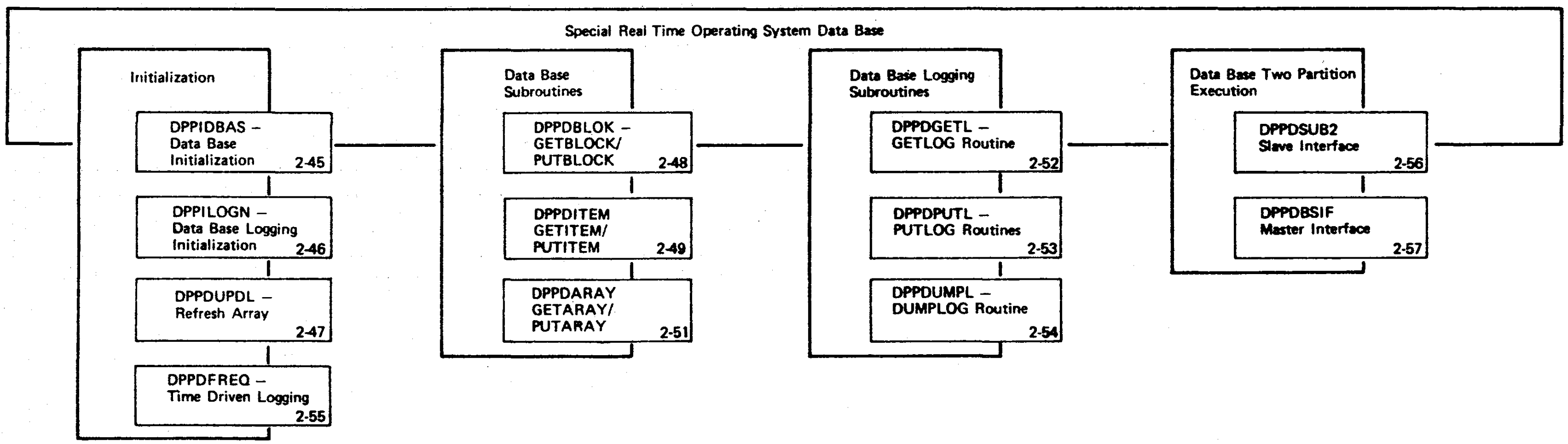

Special Real Time Operating System Data Base
Initialization
DPPIDBAS – Data Base Initialization 2-45
DPPILOGN – Data Base Logging Initialization 2-46
DPPDUPDL – Refresh Array 2-47
DPPDFREQ – Time Driven Logging 2-55
Data Base Subroutines
DPPDBLOK – GETBLOCK/ PUTBLOCK 2-48
DPPDITEM GETITEM/ PUTITEM 2-49
DPPDARAY GETARAY/ PUTARAY 2-51
Data Base Logging Subroutines
DPPDGETL – GETLOG Routine 2-52
DPPDPUTL – PUTLOG Routines 2-53
DPPDUMPL – DUMPLOG Routine 2-54
Data Base Two Partition Execution
DPPDSUB2 Slave Interface 2-56
DPPDBSIF Master Interface 2-57

Figure 2-44 - Special Real Time Operating System Data Base Overview

DPPIDBAS

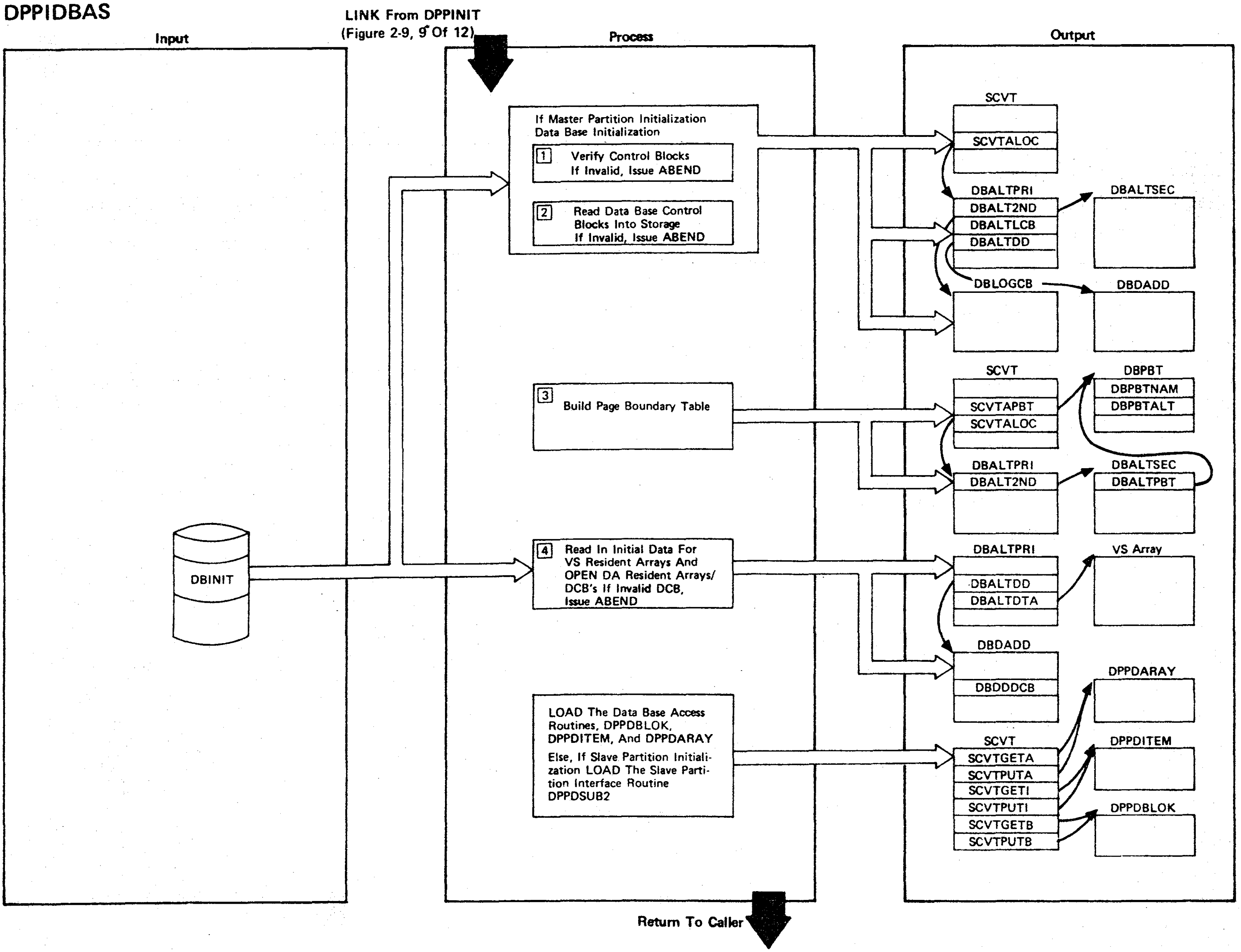


Figure 2-45 (1 Of 2) - Data Base Initialization

Figure 2-45 (2 Of 2)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If there is an invalid XCVT, SCVT, or TCBX, job step.	USER 10 USER 11	
2	The partitioned data set referenced through the DBINIT DD card contains the four primary data base control blocks (the Primary Array Locator Table, the Secondary Array Locator Table, the Data Base Logging Control Block, and the Data Base DD name table). These were built by the offline data base utility program DPPXDBIN. If there is not an @INIT member, ABEND job step. If unable to locate log array, ABEND job step. NOTE: The DBALTPRI and DBALTSEC are read into supervisor storage, and DBLOGCB and DBDADD are read into user storage.	USER 13 USER 54	DPPIDBAS DPIDBAS1
3	The DBPBT contains the name of the last array in each page of the DBALTSEC. This will allow the data access routines to locate the requested array with a minimum number of page faults to the DBALTSEC.		DPIDBAS2
4	The data for VS resident arrays is read into storage from the partitioned data set referenced through the DBINIT DD card. All data sets containing the direct access resident arrays are opened. If unable to open, ABEND job step.	USER 12	DPIDBAS3

DPPILOGN

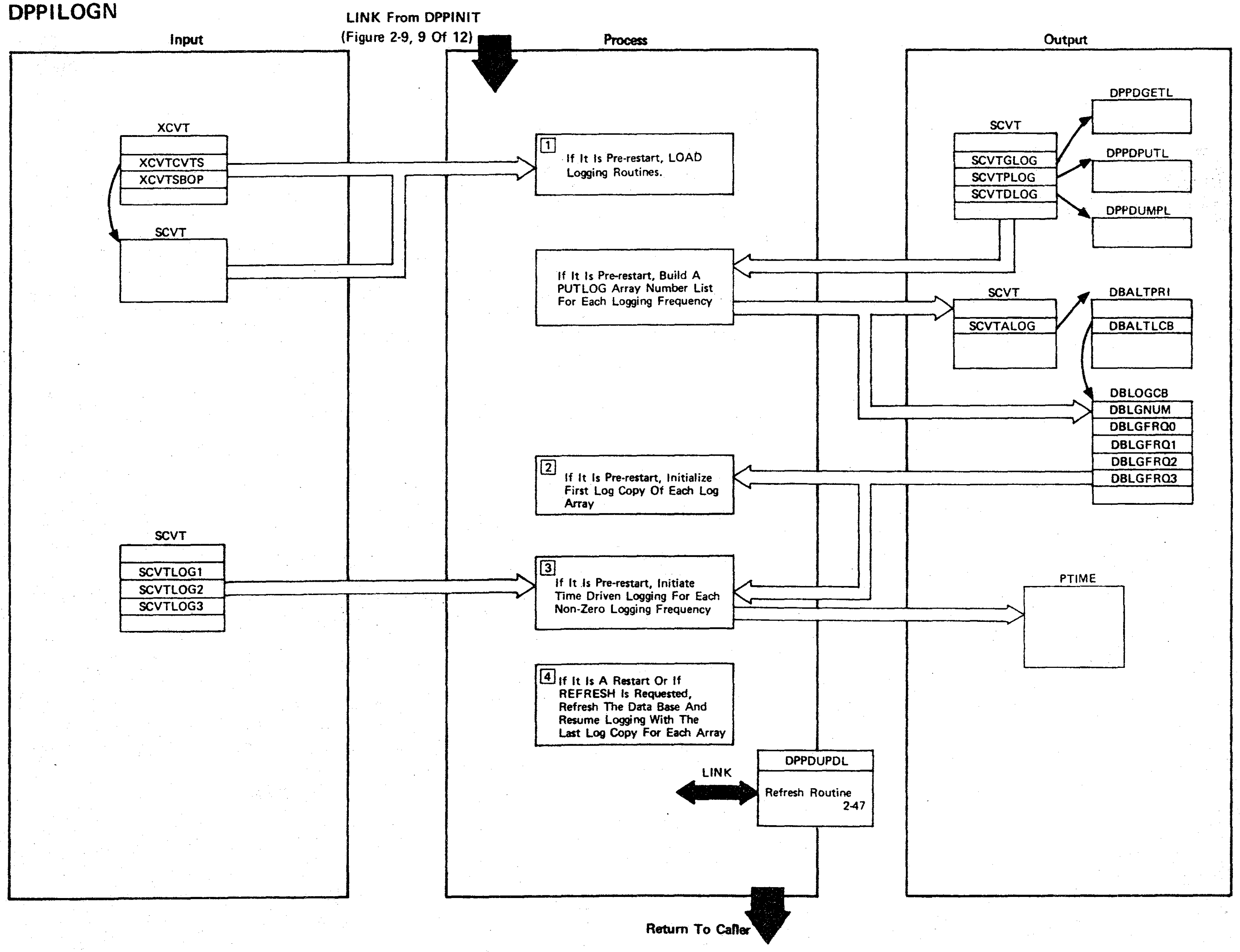

LINK From DPPINIT
(Figure 2-9, 9 Of 12)
Input
Process
Output
XCVT
XCVTCVTS
XCVTSBOP
SCVT
SCVT
SCVTLOG1
SCVTLOG2
SCVTLOG3
1 If It Is Pre-restart, LOAD Logging Routines.
If It Is Pre-restart, Build A PUTLOG Array Number List For Each Logging Frequency
2 If It Is Pre-restart, Initialize First Log Copy Of Each Log Array
3 If It Is Pre-restart, Initiate Time Driven Logging For Each Non-Zero Logging Frequency
4 If It Is A Restart Or If REFRESH Is Requested, Refresh The Data Base And Resume Logging With The Last Log Copy For Each Array
LINK
DPPDUPDL
Refresh Routine
2-47
Return To Caller
SCVT
SCVTGLOG
SCVTPLOG
SCVTDLOG
DPPDGETL
DPPDPUTL
DPPDUMPL
SCVT
SCVTALOG
DBALTPRI
DBALTLCB
DBLOGCB
DBLGNUM
DBLGFRQ0
DBLGFRQ1
DBLGFRQ2
DBLGFRQ3
PTIME

Figure 2-46 (1 Of 2) - Logging Initialization

Figure 2-46 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Logging routines DPPDGETL, DPPDPUTL, and DPPDUMPL are loaded into storage and their addresses are stored in the SCVT.		DPPILOGN
2	A PUTLOG macro call is issued for each log frequency (0, 1, 2, and 3) using the array number list as input.		DPPILOGN
3	A PTIME macro call is issued for log frequencies 1, 2, and 3 specifying TASK and EP name DPPDFREQ if the SYSGENed time interval for that particular log frequency is nonzero. Log frequency 0 is used for arrays that are to be logged on a demand basis only.		DPPILOGN
4	If an array has been defined as refreshable during the data base offline compilation, it will be refreshed from the last logged copy of that array following a restart or if data base refresh had been specified through the use of a DBREF statement in the SYSINIT input stream during a normal start. In addition, logging will be resumed with last logged copy for all logging arrays.		DPPILOGN

DPPDUPDL

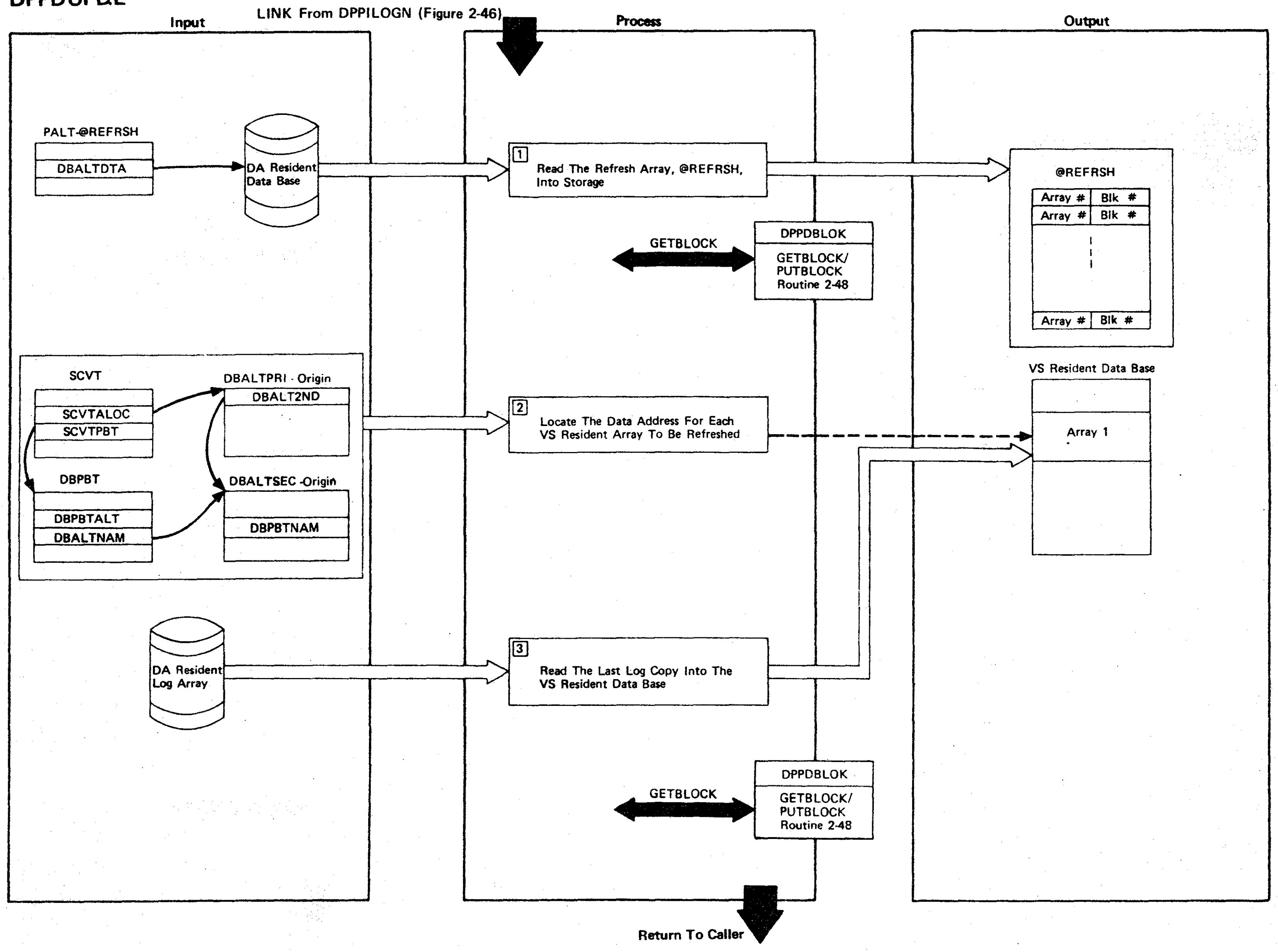


Figure 2-47 (1 Of 2) - Data Base Refresh Routine

Figure 2-47 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	A GETARRAY macro call is used to find the size and number of blocks in the refresh array, @REFRSH. Then a GETBLOCK macro call is used to retrieve the array.		DPPDUPDL
2	The Page Boundary Table and the Secondary Array Locator Table are used to calculate the array number for the named arrays. The array number is used as an index into the Primary Array Locator Table. The DBALTPRI contains the address of the VS resident array.		DPPDUPDL
3	A GETBLOCK macro call is used to read the last log copy into the VS resident data base.		DPPDUPDL

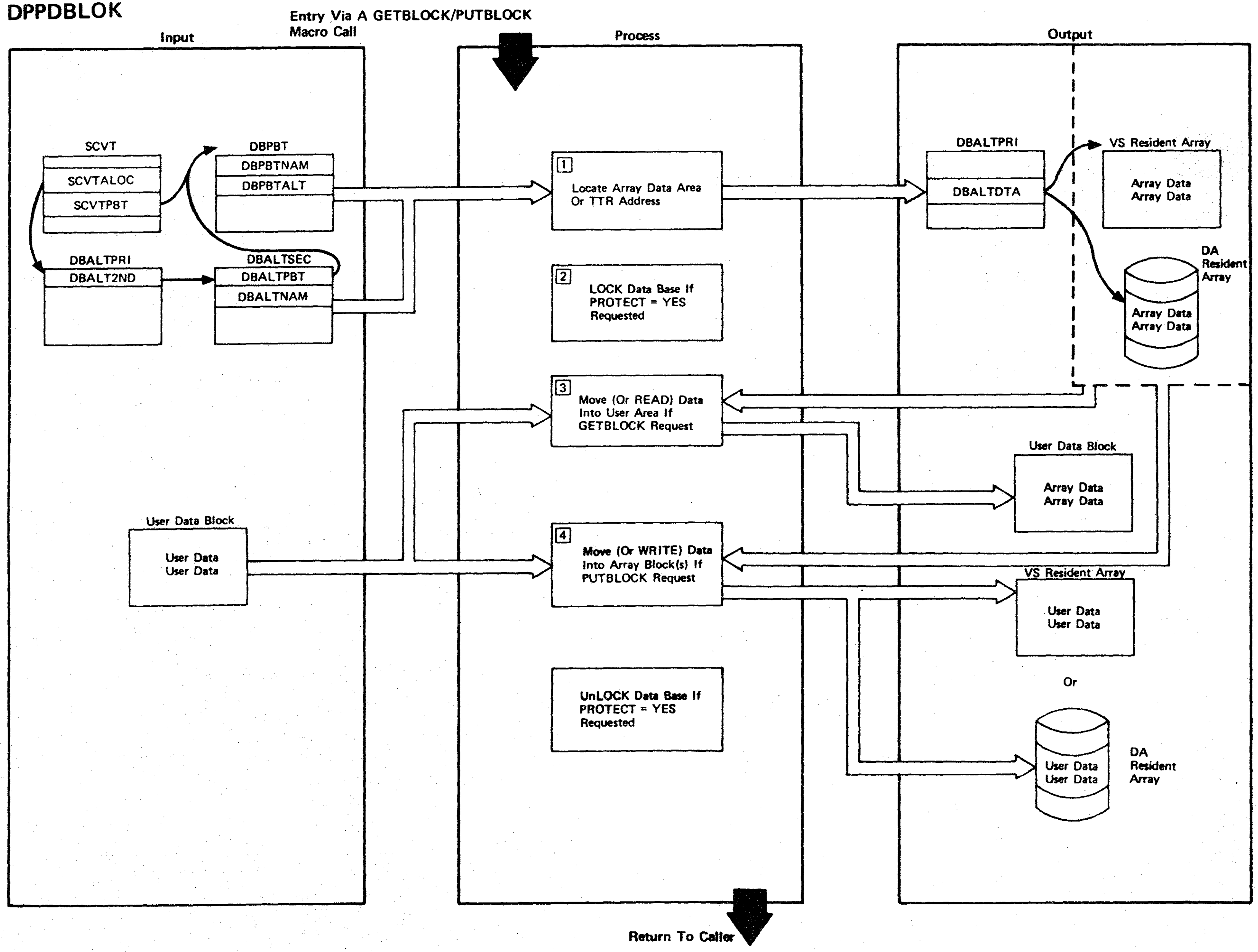


Figure 2-48 (1 Of 2) - GETBLOCK/PUTBLOCK Routine

Figure 2-48 (2 Of 2)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The DBPBT and the DBALTSEC are used to calculate the array number for the named arrays. The array number is used as an index into the DBALTPRI and DBALTPRI contains the address of the data for VS resident arrays or the relative track address of the data for DA resident arrays.		DPPDBLOK
2	If protect=YES is specified, the entire VS resident data base is locked for VS resident arrays or one Direct Access data set is locked for that DA resident array.		DPPDBLOK
3 and 4	Only the user specified blocks of the array are accessed or modified on GETBLOCK and/or PUTBLOCK requests.		DPPDBLOK

DPPDITEM

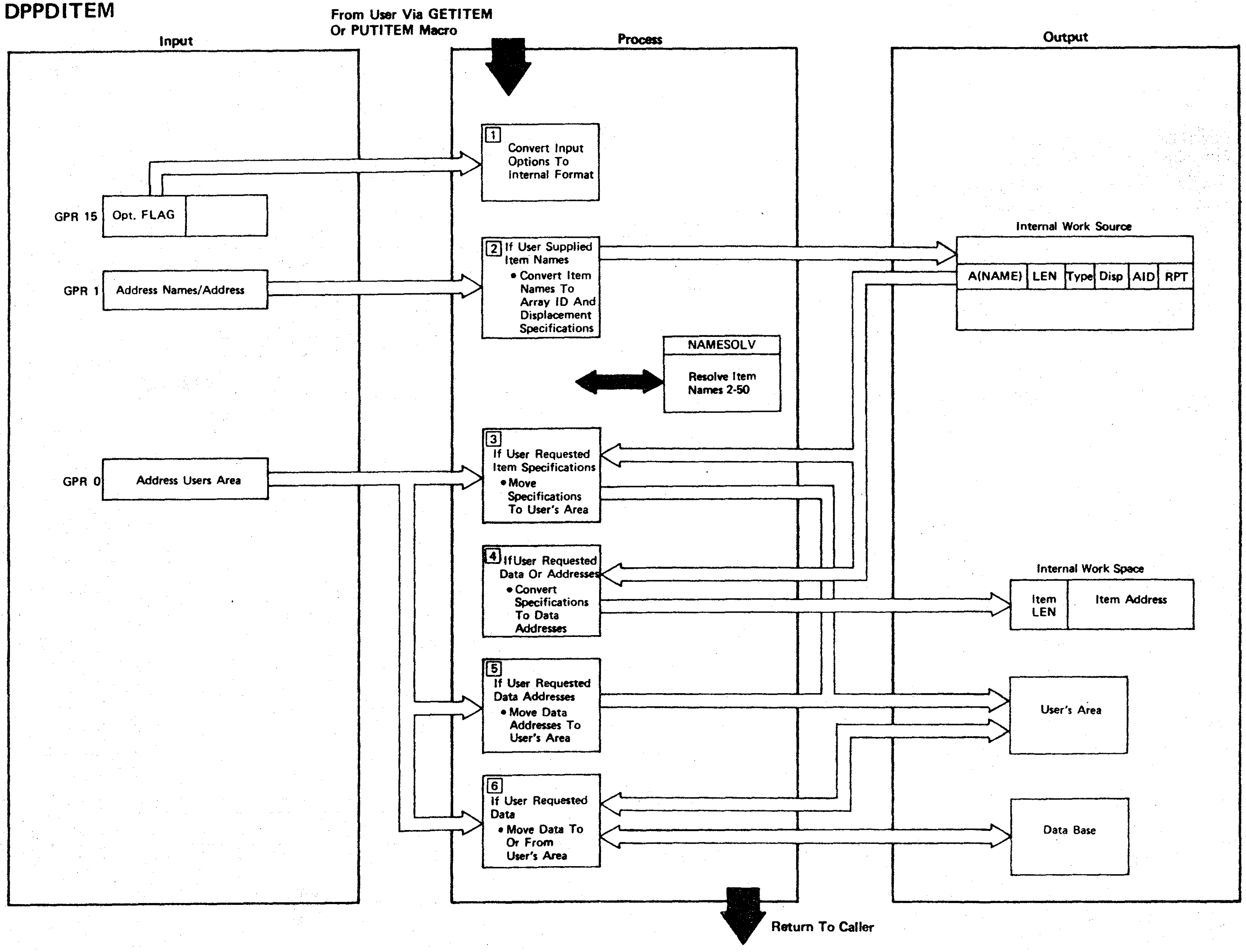

From User Via GETITEM
Or PUTITEM Macro
Input
Process
Output
GPR 15
Opt. FLAG
GPR 1
Address Names/Address
GPR 0
Address Users Area
1 Convert Input Options To Internal Format
2 If User Supplied Item Names
• Convert Item Names To Array ID And Displacement Specifications
NAMESOLV
Resolve Item Names 2-50
3 If User Requested Item Specifications
• Move Specifications To User's Area
4 If User Requested Data Or Addresses
• Convert Specifications To Data Addresses
5 If User Requested Data Addresses
• Move Data Addresses To User's Area
6 If User Requested Data
• Move Data To Or From User's Area
Internal Work Source
A(NAME)
LEN
Type
Disp
AID
RPT
Internal Work Space
Item LEN
Item Address
User's Area
Data Base
Return To Caller

Figure 2-49 (1 Of 2) - Access Data Base Items

Figure 2-49 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	This program may be entered for GETITEM or PUTITEM processing. The user may supply the name of a single item for which data is to be processed or a list of items.		DPPDITEM
2	Program segment NAMESOLV converts item names to internal format.		DPPDITEM
3	Program segment MOVESPEC will move the specification data to the user's area.		DPPDITEM
4	Program segment CONVSPEC will convert the specification data to addresses.		DPPDITEM
5	Program segment MOVEADDR will move the item addresses to the user's area.		DPPDITEM
6	Program segment MOVEDATA will move the item data to the user's area if a GETITEM is being processed or from the user's area to the data base if a PUTITEM is being processed.		DPPDITEM

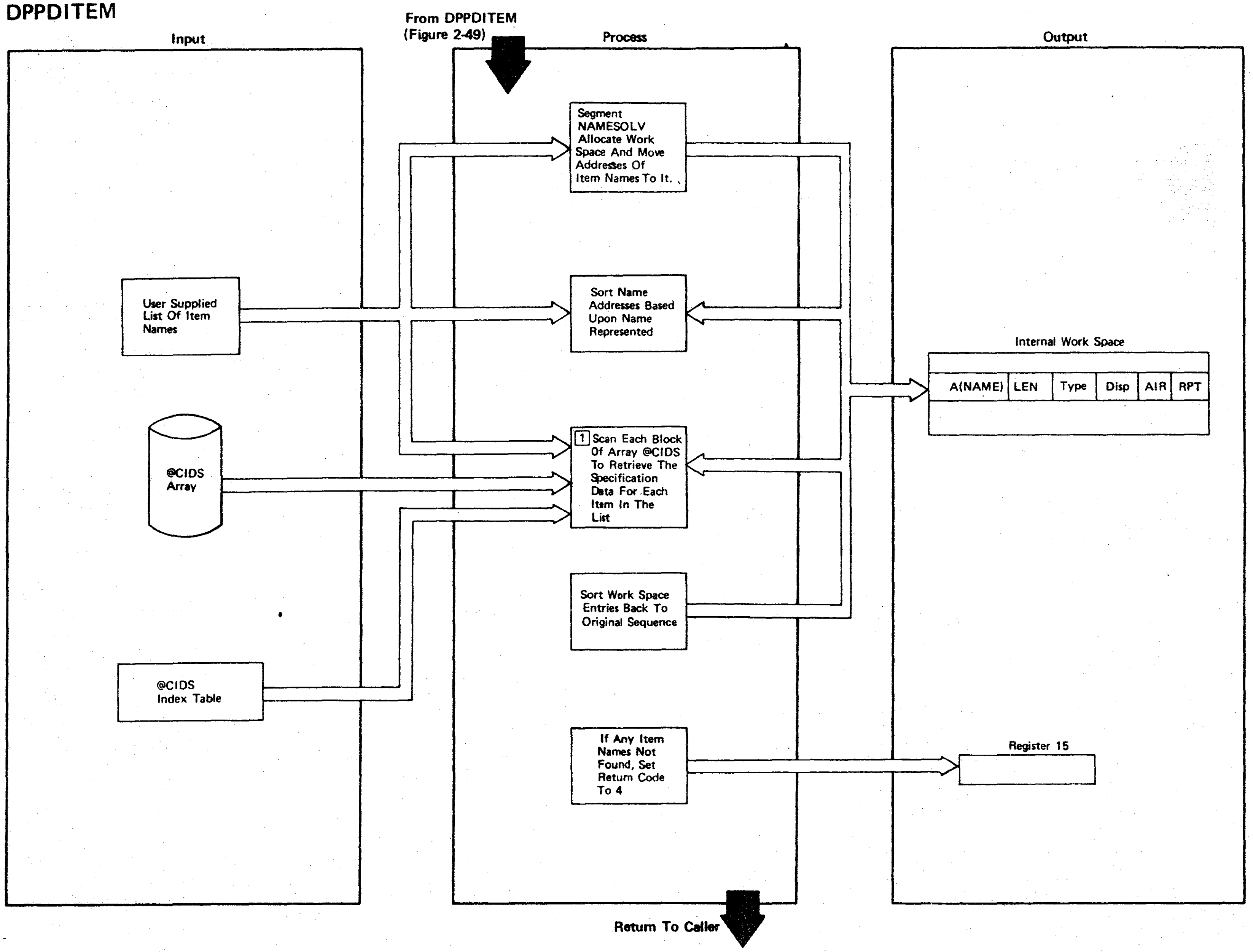

From DPPDITEM
(Figure 2-49)
Input
Process
Output
User Supplied List Of Item Names
@CIDS Array
@CIDS Index Table
Segment NAMESOLV Allocate Work Space And Move Addresses Of Item Names To It.
Sort Name Addresses Based Upon Name Represented
1 Scan Each Block Of Array @CIDS To Retrieve The Specification Data For Each Item In The List
Sort Work Space Entries Back To Original Sequence
If Any Item Names Not Found, Set Return Code To 4
Internal Work Space
A(NAME)
LEN
Type
Disp
AIR
RPT
Register 15
Return To Caller

Figure 2-50 (1 Of 2) - Access Data Base Items (NAMESOLV)

Figure 2-50 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The array named @CIDS contains the item specifications for every item defined to the system.		DPPDITEM
2	The @CIDS index table is used to locate the block of the DA resident array that contains the specified item name, if it exists. This reduces the number of I/O operators required to locate this item name.		

DPPDARAY

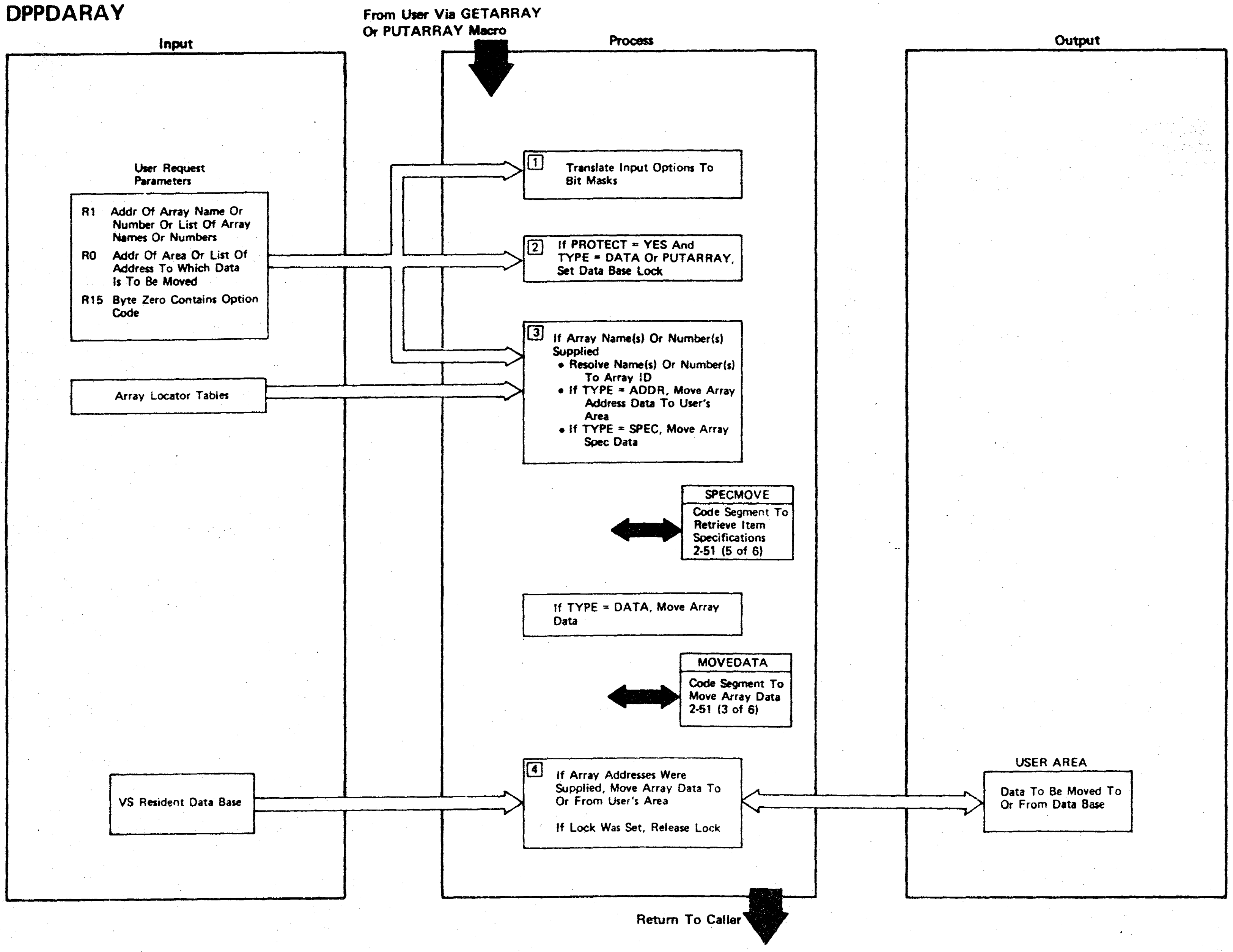

From User Via GETARRAY
Or PUTARRAY Macro
Input
Process
Output
User Request
Parameters
R1 Addr Of Array Name Or Number Or List Of Array Names Or Numbers
R0 Addr Of Area Or List Of Address To Which Data Is To Be Moved
R15 Byte Zero Contains Option Code
Array Locator Tables
VS Resident Data Base
1 Translate Input Options To Bit Masks
2 If PROTECT = YES And TYPE = DATA Or PUTARRAY, Set Data Base Lock
3 If Array Name(s) Or Number(s) Supplied
• Resolve Name(s) Or Number(s) To Array ID
• If TYPE = ADDR, Move Array Address Data To User's Area
• If TYPE = SPEC, Move Array Spec Data
SPECMOVE
Code Segment To Retrieve Item Specifications 2-51 (5 of 6)
If TYPE = DATA, Move Array Data
MOVEDATA
Code Segment To Move Array Data 2-51 (3 of 6)
4 If Array Addresses Were Supplied, Move Array Data To Or From User's Area
If Lock Was Set, Release Lock
USER AREA
Data To Be Moved To Or From Data Base
Return To Caller

Figure 2-51 (1 Of 6) - GETARRAY/PUTARRAY Processor

Figure 2-51 (2 of 6).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The processing for GETARRAY or PUTARRAY is identical with the exception of the direction of the data movement if TYPE=DATA.		DPPDARAY
2	The protect function disallows a user from moving data into or out of the data base until previous users have completed their data moves.		DPPDARAY
3	The user may specify a single array name, number or address or a list of array names, numbers or addresses. In either case, the processing is similar with the exception that the program loops through this logic if lists are supplied.		DPPDARAY
4	Array addresses may be resolved and passed to this program to bypass name resolution on each use of a list. The addresses passed are used as the address of the array.		DPPDARAY

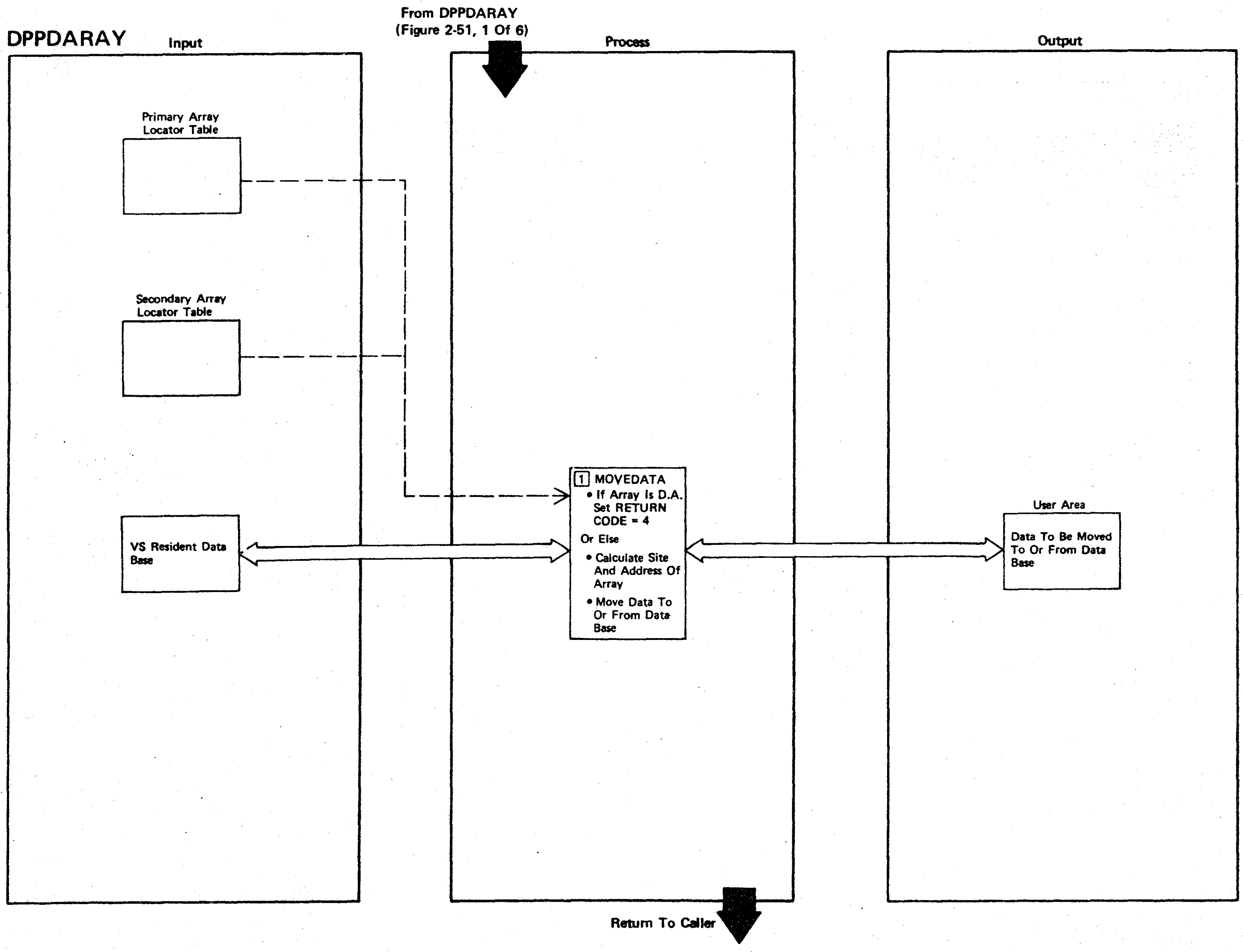

DPPDARAY
Input
From DPPDARAY
(Figure 2-51, 1 Of 6)
Process
Output
Primary Array Locator Table
Secondary Array Locator Table
VS Resident Data Base
1 MOVEDATA
• If Array Is D.A. Set RETURN CODE = 4
Or Else
• Calculate Site And Address Of Array
• Move Data To Or From Data Base
User Area
Data To Be Moved To Or From Data Base
Return To Caller

Figure 2-51 (3 Of 6) - GETARRAY/PUTARRAY Processor (MOVEDATA)

Figure 2-51 (4 of 6).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	GETARRAY and PUTARRAY processing is identical with the exception of the direction of the movement of data if TYPE=DATA.		DPPDARAY

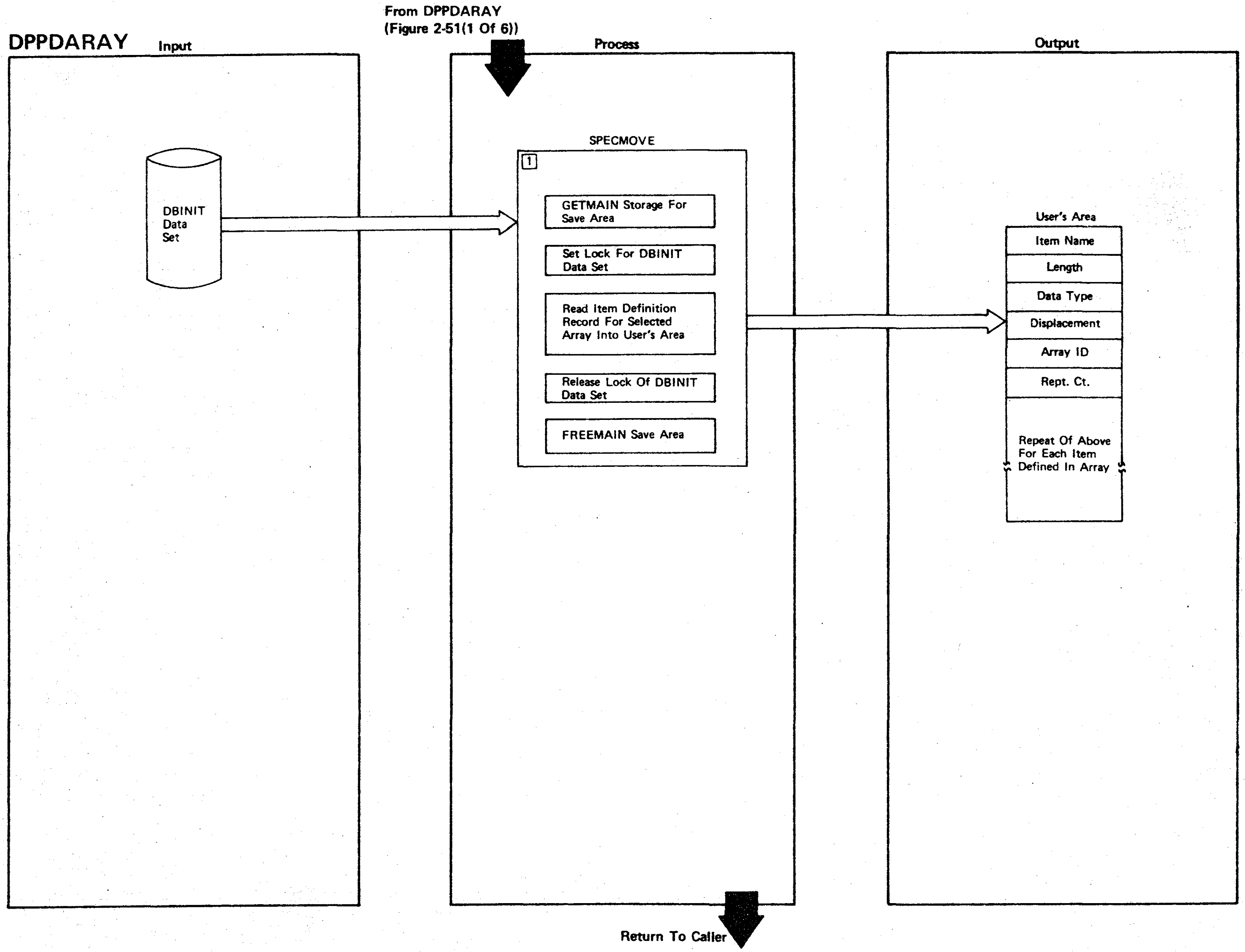

DPPDARAY
Input
From DPPDARAY
(Figure 2-51(1 Of 6))
Process
Output
DBINIT
Data
Set
SPECMOVE
1
GETMAIN Storage For Save Area
Set Lock For DBINIT Data Set
Read Item Definition Record For Selected Array Into User's Area
Release Lock Of DBINIT Data Set
FREEMAIN Save Area
User's Area
Item Name
Length
Data Type
Displacement
Array ID
Rept. Ct.
Repeat Of Above For Each Item Defined In Array
Return To Caller

Figure 2-51 (5 Of 6) - GETARRAY/PUTARRAY Processor (SPECMOVE)

Figure 2-51 (6 of 6).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The DBINIT data set contains a logical record for each array which consists of a 16-byte entry for each item defined for the array.		DPPDARAY

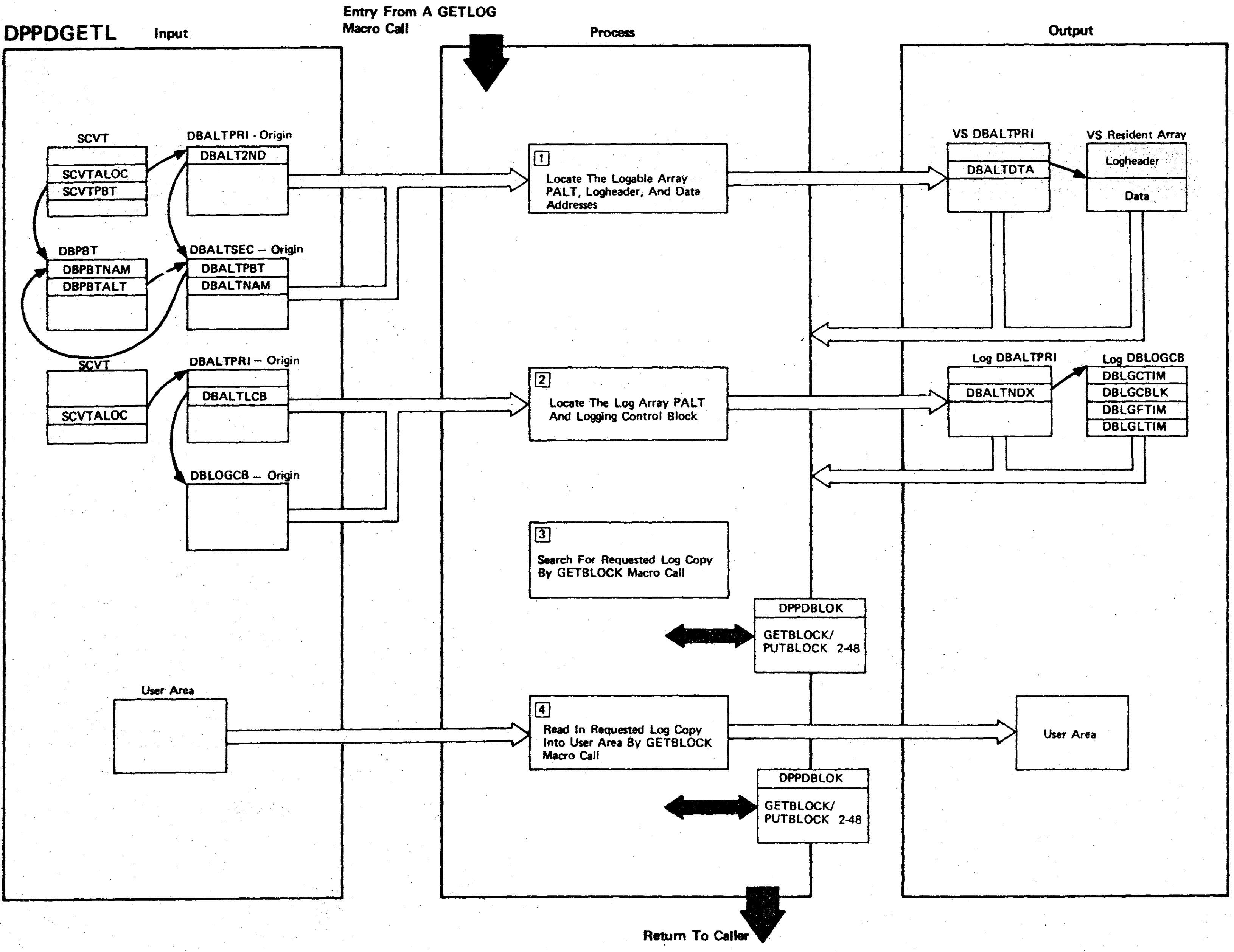
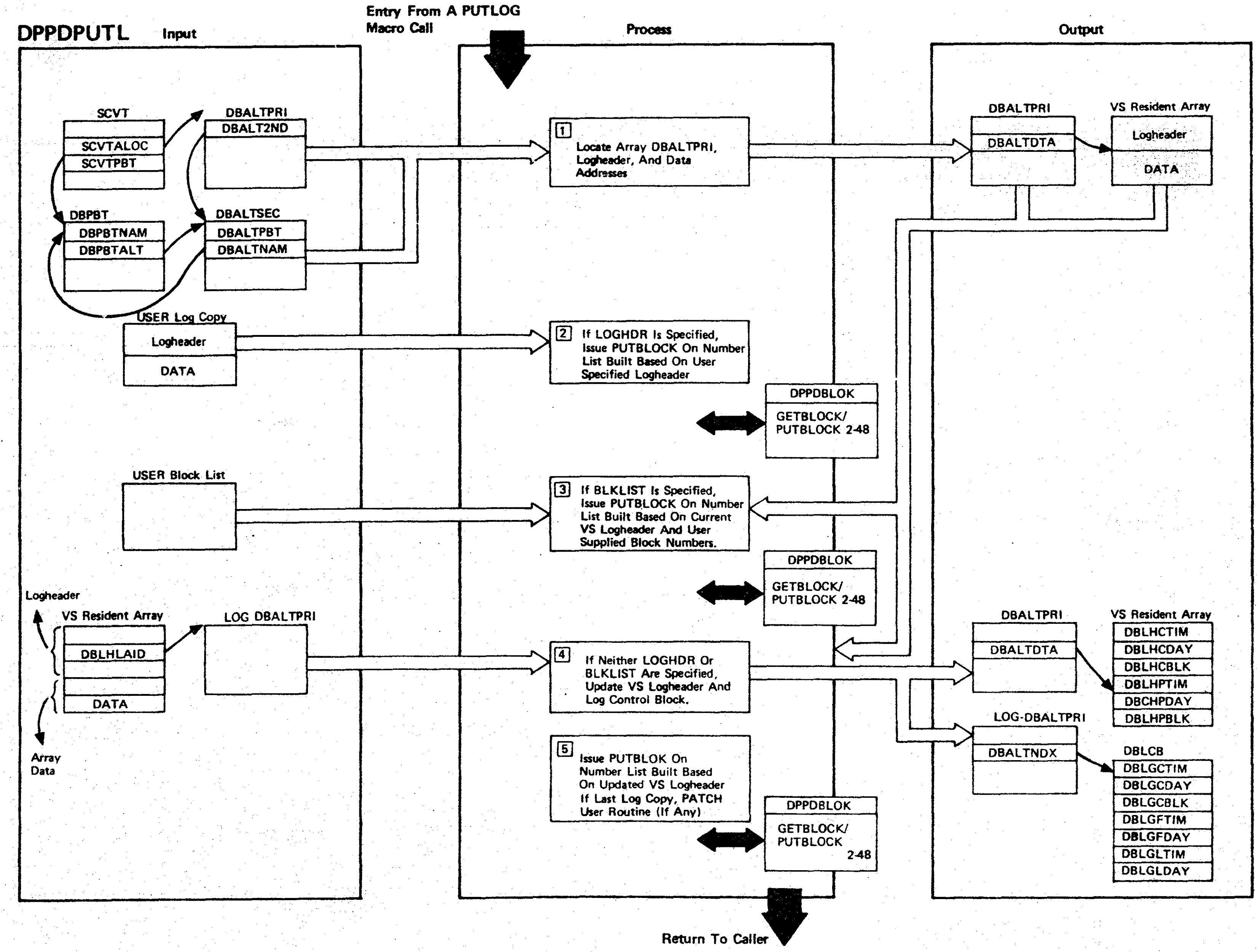


Figure 2-52 (1 Of 2) - GETLOG Routine

Figure 2-52 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The DBPBT and the DBALTSEC are used to calculate the array number for the named arrays. The array number is used as an index into the DBALTPRI. The DBALTPRI contains the address of the VS resident array. The logheader precedes the array data in storage.		DPPDGETL
2	The VS array logheader contains the array number for the associated log array. The log array number is used as an index into the DBALTPRI. The DBALTPRI for the loggable array contains an index into the data base logging control block.		DPPDGETL
3	The time fields in the DBLOGCB are used to reduce the search for the requested log copy. GETBLOCK subroutine is used to read the log-header from the log copies until the correct log copy is found.		DPPDGETL
4	GETBLOCK subroutine is used to read the entire log copy into the user provided area.		DPPDGETL



1. Locate Array DBALTPRI, Logheader, And Data Addresses

2. If LOGHDR Is Specified, Issue PUTBLOCK On Number List Built Based On User Specified Logheader

3. If BLKLIST Is Specified, Issue PUTBLOCK On Number List Built Based On Current VS Logheader And User Supplied Block Numbers.

4. If Neither LOGHDR Or BLKLIST Are Specified, Update VS Logheader And Log Control Block.

5. Issue PUTBLOK On Number List Built Based On Updated VS Logheader If Last Log Copy, PATCH User Routine (If Any)

Figure 2-53 (1 Of 2) - PUTLOG Routine

Figure 2-53 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The DBPBT and the DBALTSEC are used to calculate the array number for the named arrays. The array number is used as an index into the DBALTPRI. The DBALTPRI contains the address of the VS resident array. The logheader precedes the array data in storage.		DPPDPUTL
2	The LOGHDR option is used to replace a log copy in the log array.		DPPDPUTL
3	The BLKLIST option is used to update blocks within the current log copy of the Log Array.		DPPDPUTL
4	The logheader and Log Control Block are modified to point to the next log copy.		DPPDPUTL
5	Normal logging copies the current VS resident array and its log header into the next log copy. If it is determined that wraparound will occur with the next PUTLOG request, the user defined wraparound processor (if any) is patched.		DPPDPUTL

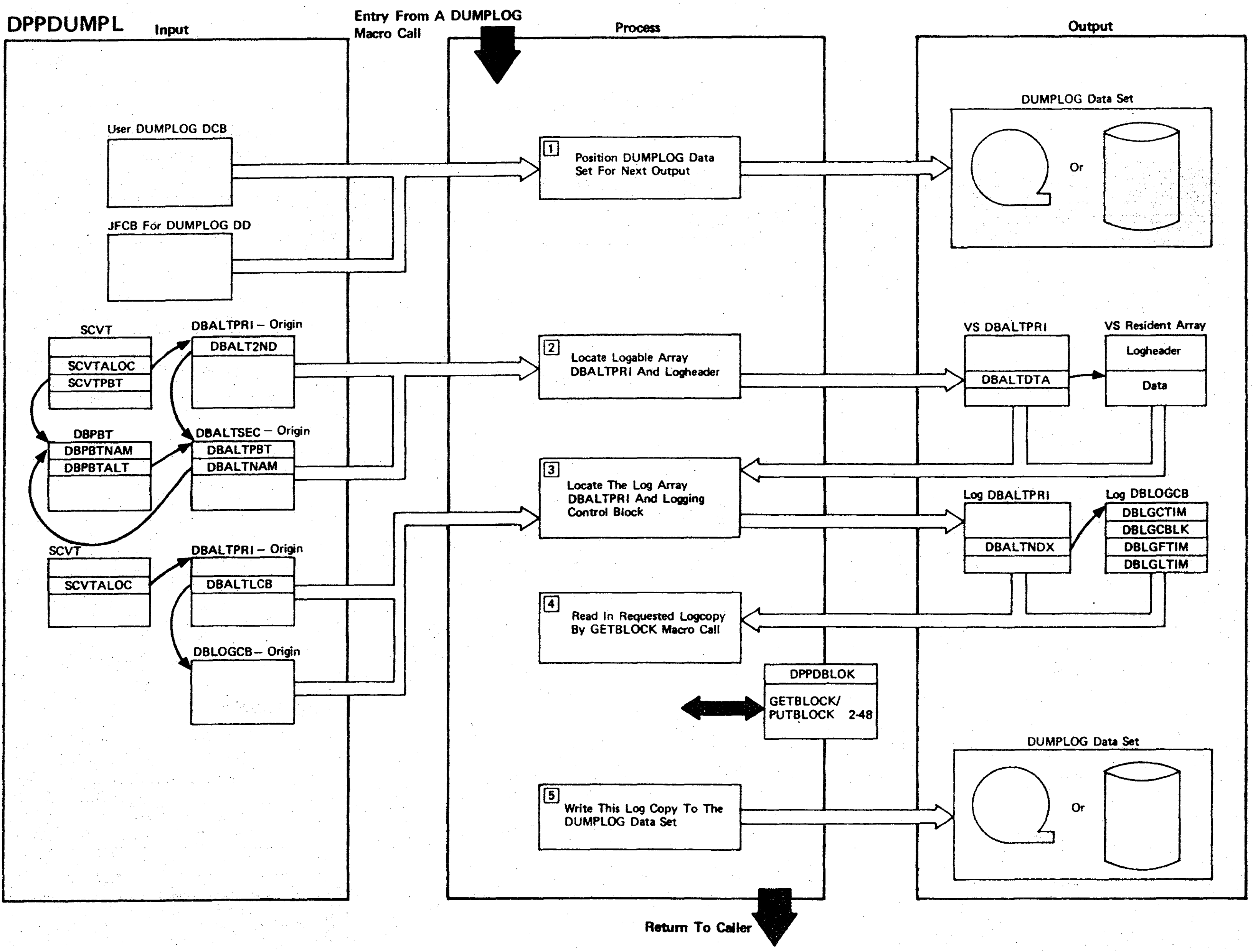


Figure 2-54 (1 Of 2) - DUMPLOG Routine

Figure 2-54 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The user supplied DD name defines the DUMPLOG data set. This data set is opened and positioned according to the options specified on the DUMPLOG macro.		DPPDUMPL
2	The DBPBT and the DBALTSEC are used to calculate the array number for named arrays. The array numbers used as an index into the DBALTPRI. The DBALTPRI contains the address of the data for the VS resident array, and the logheader immediately precedes the array data in storage.		DPPDUMPL
3	The VS array logheader contains the array number for the associated log array. The log array number is used as an index into the DBALTPRI. The DBALTPRI for the loggable array contains an index into the data base logging control block.		DPPDUMPL
4	The requested log copy is read into VS storage.		
5	The log copy is then written to the DUMPLOG data set using variable blocked spanned records.		

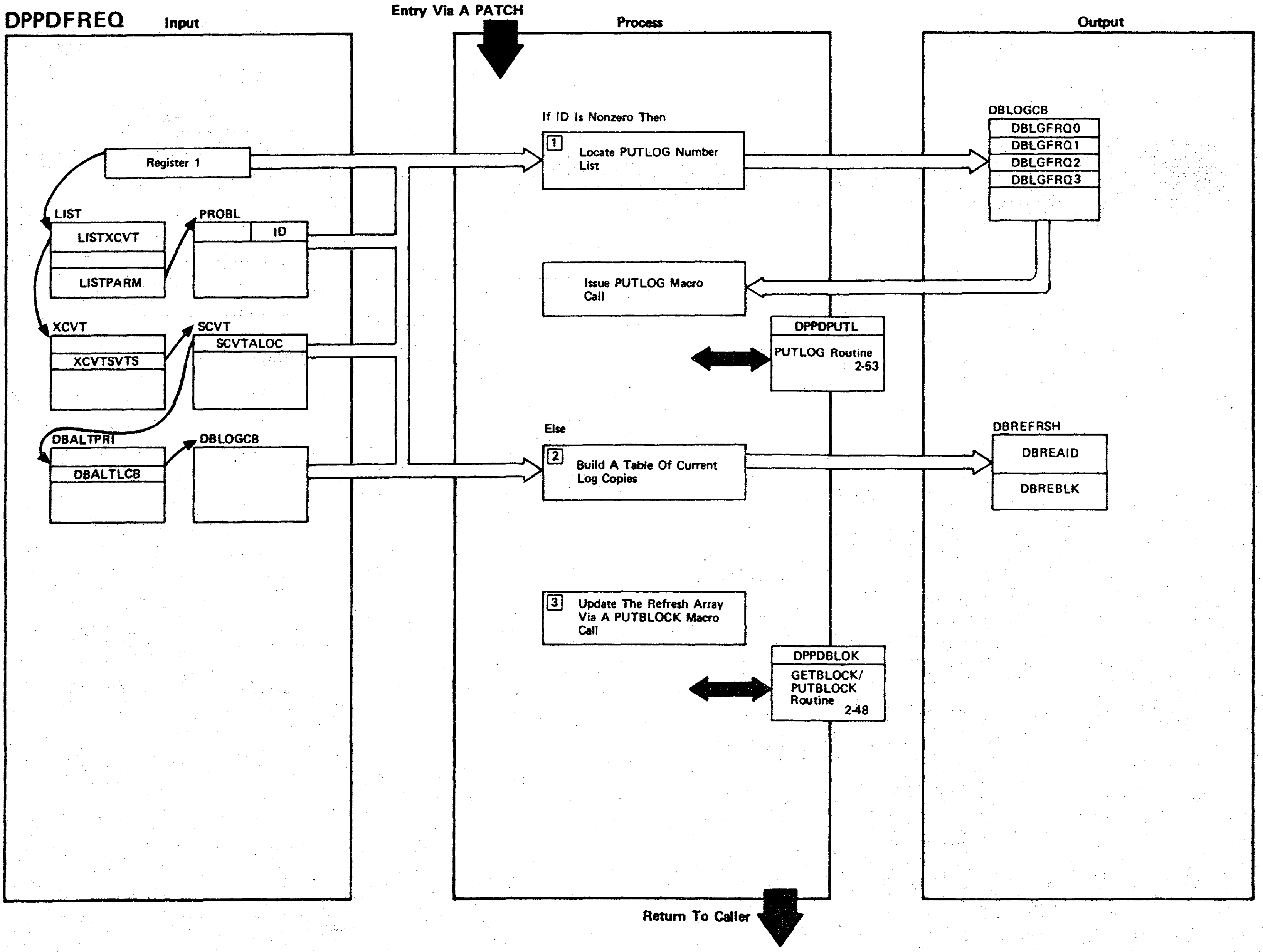
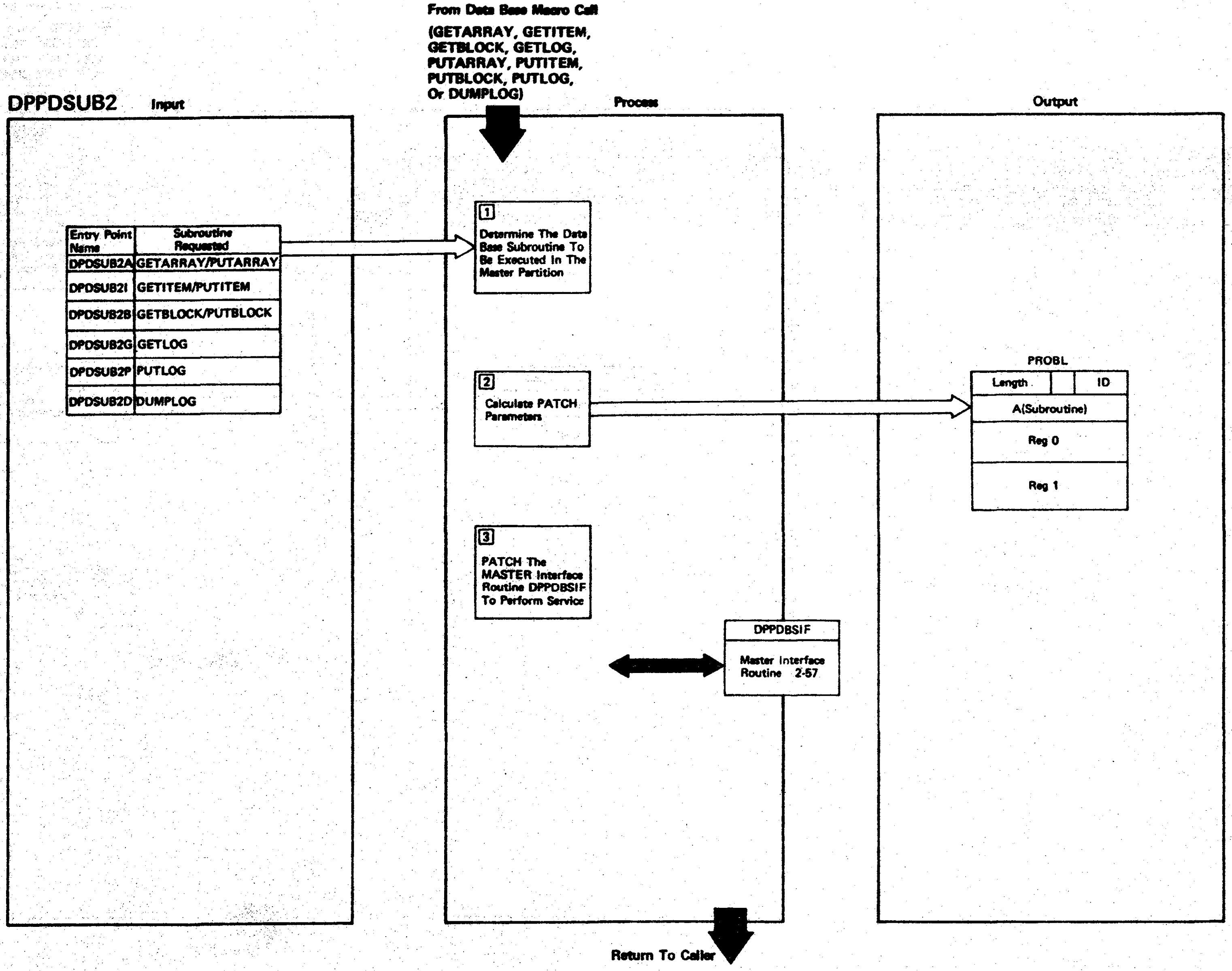


Figure 2-55 (1 Of 2) - Time Driven Logging

Figure 2-55 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The ID passed through the PATCH macro (4, 8, or 12) is used as an index into the table of PUTLOG number lists contained in the Log Control Block. The patch is the result of the PTIME SVCs issued by DPPILOGN during initialization.		DPPDFREQ
2	The DBLOGCB tables are used to build the DBREFRSH table which contains the current block number for each log array.		DPPDFREQ
3	GETBLOCK macro call is used to write the DBREFRSH table into the refresh array, @REFRSH.		DPPDFREQ



From Data Base Macro Call
(GETARRAY, GETITEM, GETBLOCK, GETLOG, PUTARRAY, PUTITEM, PUTBLOCK, PUTLOG, Or DUMPLOG)
DPPDSUB2
Input
Process
Output
1 Determine The Data Base Subroutine To Be Executed In The Master Partition
2 Calculate PATCH Parameters
3 PATCH The MASTER Interface Routine DPPDBSIF To Perform Service
DPPDBSIF
Master Interface Routine 2-57
PROBL
Length
ID
A(Subroutine)
Reg 0
Reg 1
Return To Caller

Entry Point Name	Subroutine Requested
DPDSUB2A	GETARRAY/PUTARRAY
DPDSUB2I	GETITEM/PUTITEM
DPDSUB2B	GETBLOCK/PUTBLOCK
DPDSUB2G	GETLOG
DPDSUB2P	PUTLOG
DPDSUB2D	DUMPLOG

Figure 2-56 (1 Of 2) - Data Base SLAVE Interface

Figure 2-56 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Each macro call in the SLAVE partition branches to a unique location in load module DPPDSUB2. The location is used to determine correct subroutine to be executed in the MASTER partition.		DPPDSUB2
2	The PATCH parameter list consists of the address (in the MASTER partition) of the data base subroutine to be executed and registers 0 and 1 as passed to DPPDSUB2 by the user as the result of the macro call.		DPPDSUB2
3	DPPDBSIF is patched in the MASTER partition to branch to the subroutine. DPPDSUB2 WAITS until the PATCH has completed.		DPPDSUB2

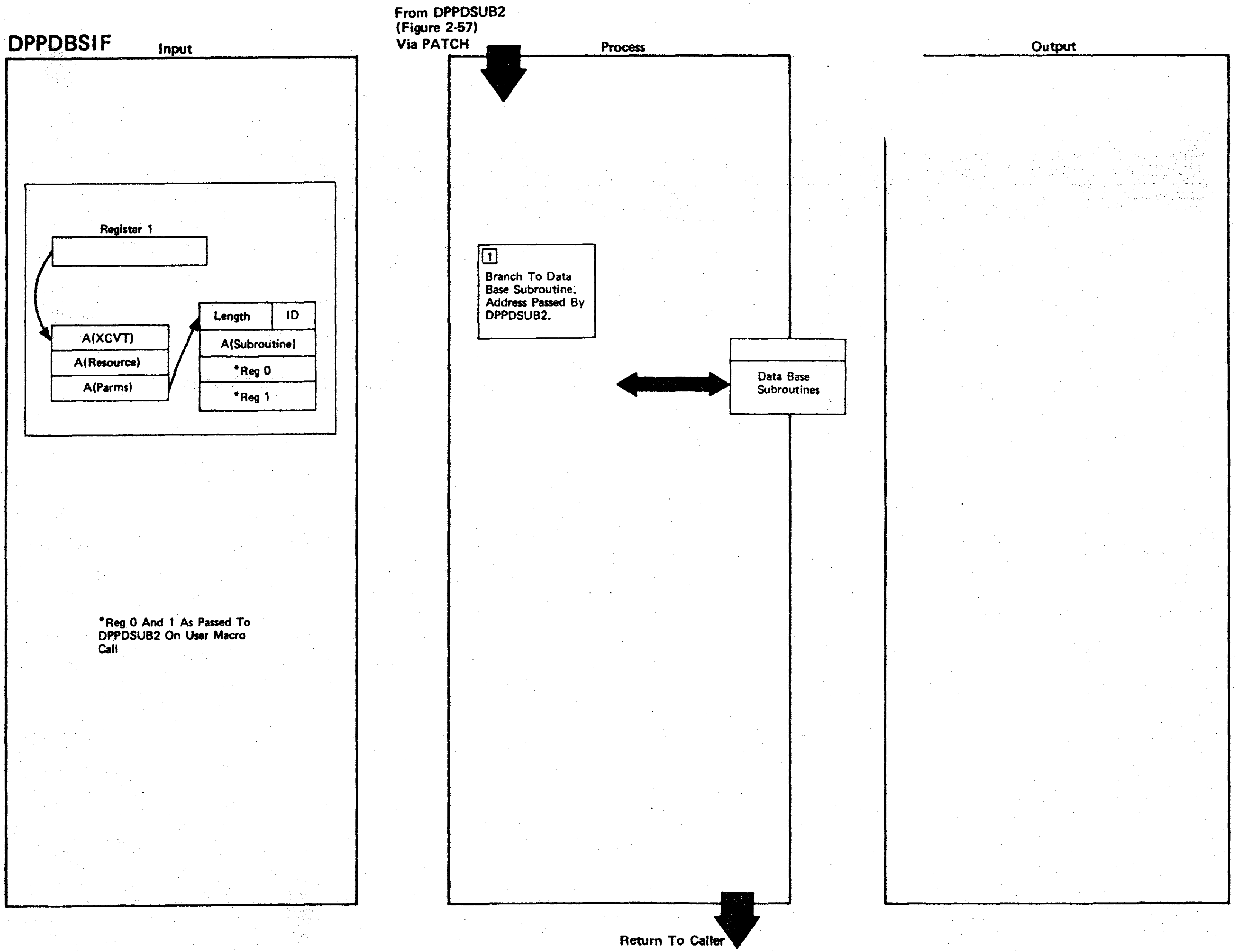


Figure 2-57 (1 Of 2) - Data Base MASTER Interface

Figure 2-57 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The address of the data base subroutine to be executed is passed as a parameter.		DPPDBSIF

Message Handler

The Special Real Time Operating System's message handler interfaces with the user either through a MESSAGE macro call or an MSGRC operator command.

The MESSAGE macro processor, DPPMMSG, is responsible for validating the input parameters from the MESSAGE macro call and formatting the actual message. The message handler output routine, DPPMMSGI, is patched by DPPMMSG and is responsible for output of the message to the appropriate device(s).

The message routing code STATUS/CHANGE routine gains control from the input message processor via a PATCH macro call as the result of a MSGRC operator command.

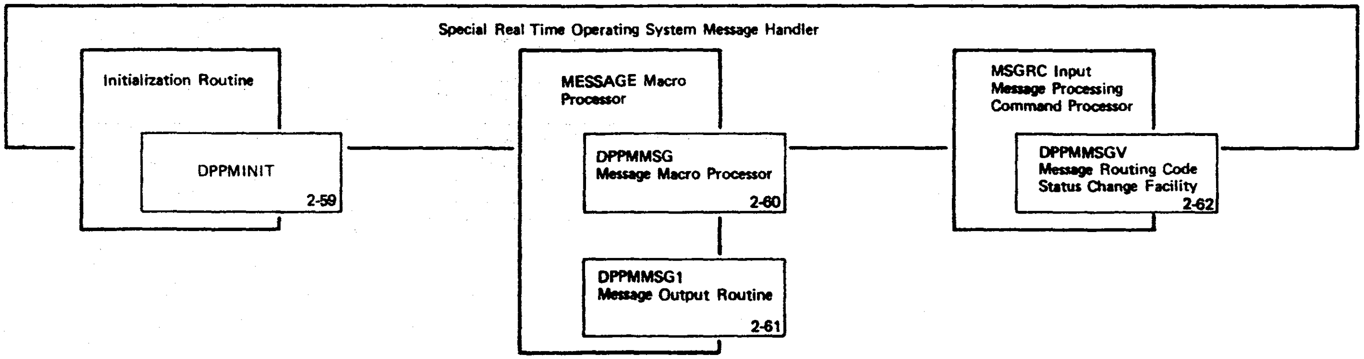

Special Real Time Operating System Message Handler
Initialization Routine
DPPMINIT
2-59
MESSAGE Macro Processor
DPPMMSG
Message Macro Processor
2-60
DPPMMSG1
Message Output Routine
2-61
MSGRC Input Message Processing Command Processor
DPPMMSGV
Message Routing Code Status Change Facility
2-62

Figure 2-58 - Special Real Time Operating System Message Handler Overview

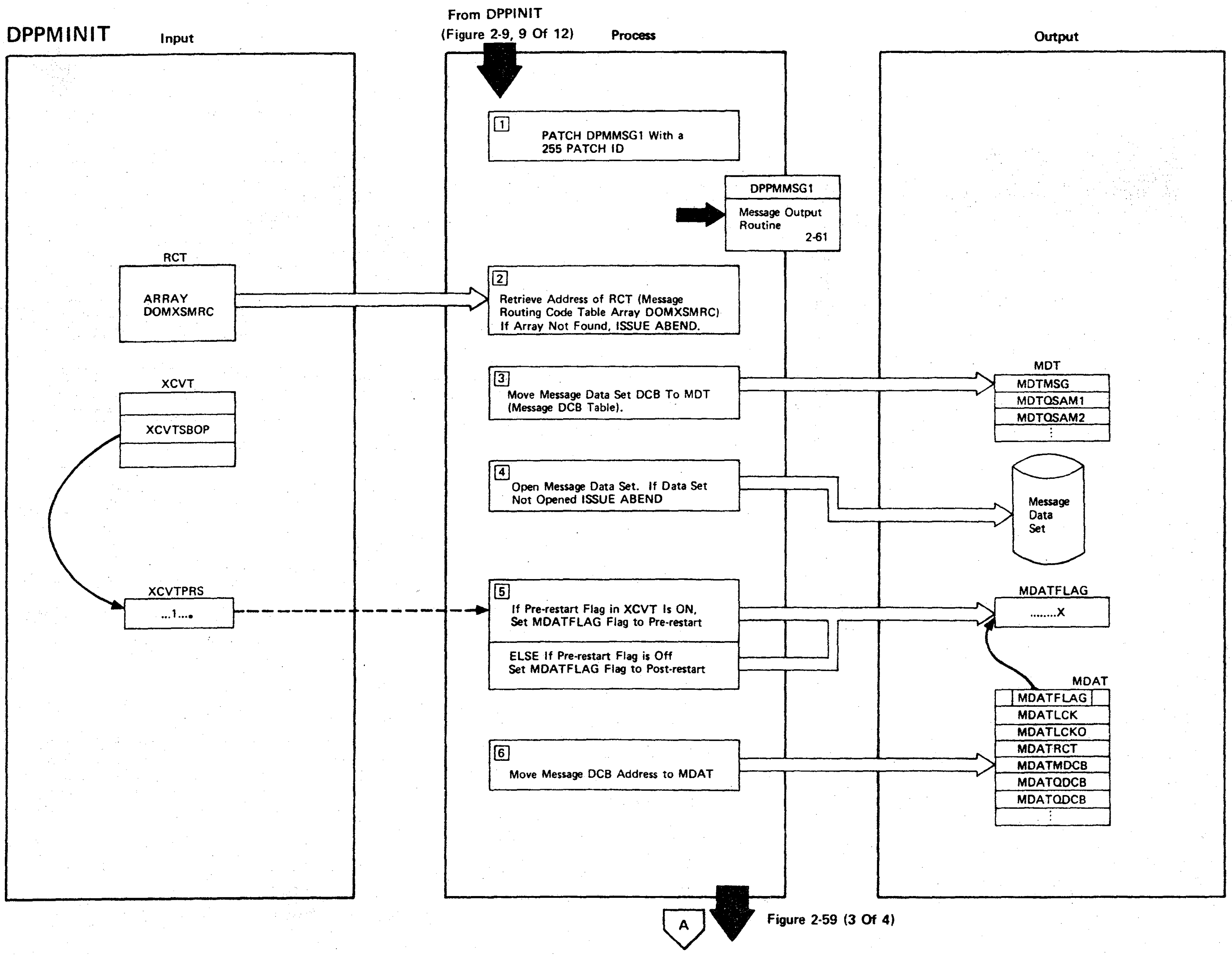

DPPMINIT
Input
From DPPINIT
(Figure 2-9, 9 Of 12)
Process
Output
1
PATCH DPMMSG1 With a 255 PATCH ID
DPPMMSG1
Message Output Routine
2-61
RCT
ARRAY DOMXSMRC
2
Retrieve Address of RCT (Message Routing Code Table Array DOMXSMRC) If Array Not Found, ISSUE ABEND.
XCVT
XCVTSBOP
3
Move Message Data Set DCB To MDT (Message DCB Table).
MDT
MDTMSG
MDTQSAM1
MDTQSAM2
4
Open Message Data Set. If Data Set Not Opened ISSUE ABEND
Message Data Set
XCVTPRS
...1...
5
If Pre-restart Flag in XCVT Is ON, Set MDATFLAG Flag to Pre-restart
ELSE If Pre-restart Flag is Off Set MDATFLAG Flag to Post-restart
MDATFLAG
........X
MDAT
MDATFLAG
MDATLCK
MDATLCKO
MDATRCT
MDATMDCB
MDATQDCB
MDATQDCB
6
Move Message DCB Address to MDAT
A
Figure 2-59 (3 Of 4)

Figure 2-59 (1 Of 4) - Message Handler Initialization

Figure 2-59 (2 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The message handler initialization routine, DPPMINIT, will create a Special Real Time Operating System task with task name DPPMMSG1 for the message output task (DPPMMSG1).		DPPMINIT
2	The address of the message routing code table (array DOMXSMRC) will be obtained from the data base. If the array cannot be found, ABEND 23 will be issued.	USER 23	DPPMINIT
3	The contents of the message data set will be moved to the message DCB table (MDT).		DPPMINIT
4	The message data set pointed to by the MSGDS DD card will be opened as a BPAM input data set. If the data set cannot be opened, ABEND 20 will be issued.	USER 20	DPPMINIT
5	The MDAT flag is a one bit flag in the message address table (MDAT). This flag will be turned off (set to 0) if the pre-restart flag (XCVTSBOP) in the XCVT is on. The flag will be turned on (set to 1) if XCVTSBOP is post-restart.		DPPMINIT
6	The address of the message DCB will be placed in the MDAT.		DPPMINIT

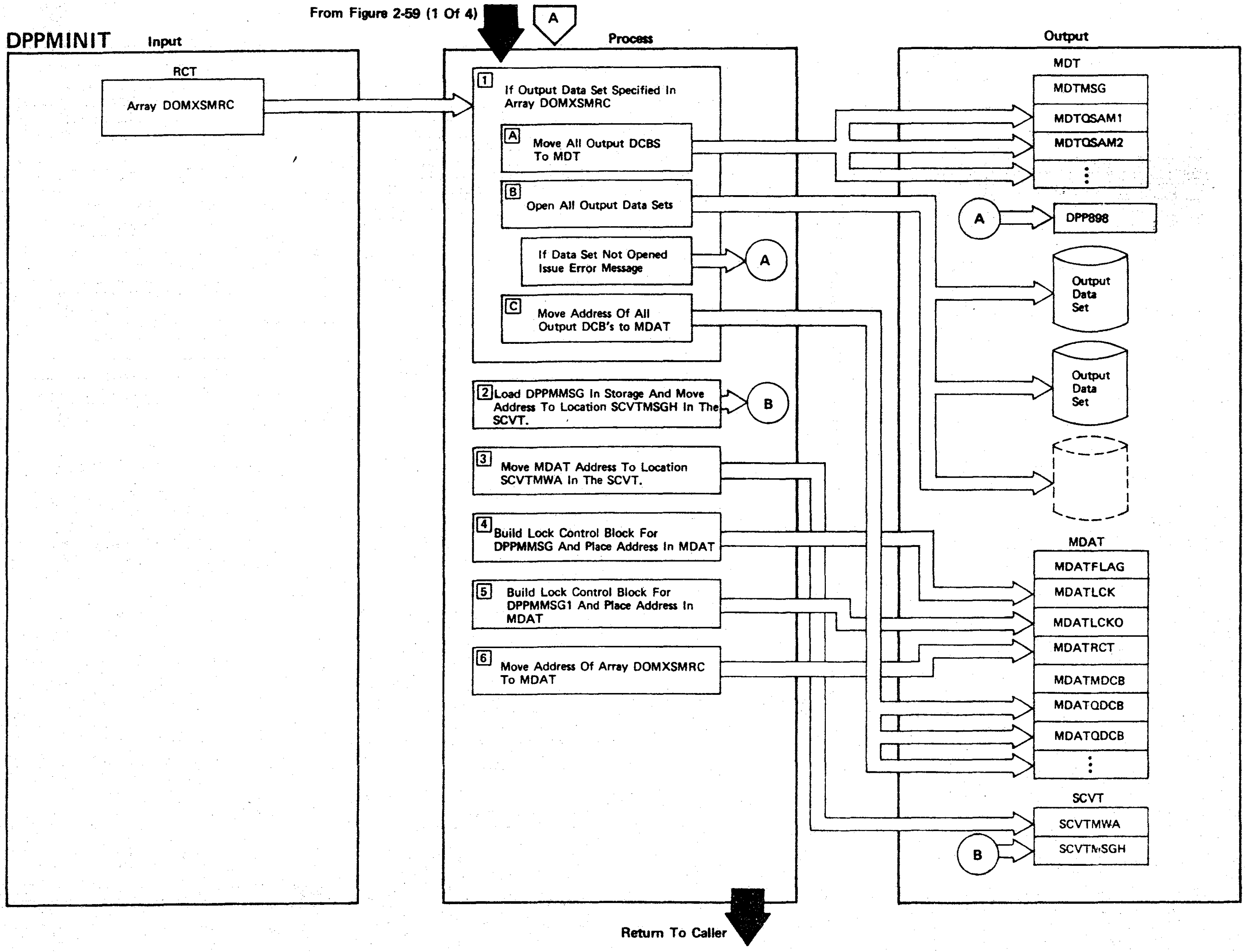
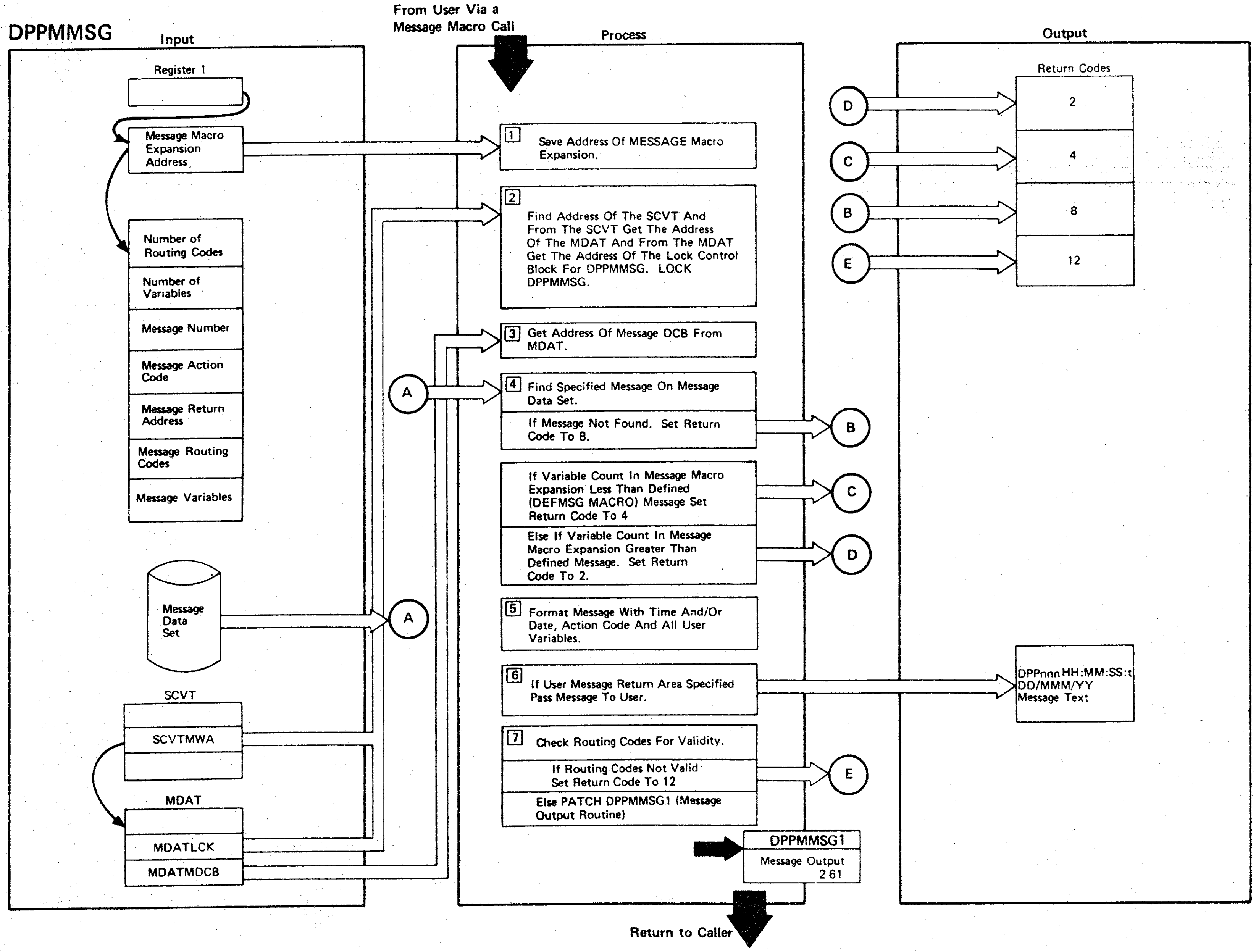

DPPMINIT
Input
Process
Output
From Figure 2-59 (1 Of 4)
A
RCT
Array DOMXSMRC
1 If Output Data Set Specified In Array DOMXSMRC
A Move All Output DCBS To MDT
B Open All Output Data Sets
If Data Set Not Opened Issue Error Message
A
C Move Address Of All Output DCB's to MDAT
2 Load DPPMMSG In Storage And Move Address To Location SCVTMSGH In The SCVT.
B
3 Move MDAT Address To Location SCVTMWA In The SCVT.
4 Build Lock Control Block For DPPMMSG And Place Address In MDAT
5 Build Lock Control Block For DPPMMSG1 And Place Address In MDAT
6 Move Address Of Array DOMXSMRC To MDAT
Return To Caller
MDT
MDTMSG
MDTQSAM1
MDTQSAM2
A
DPP898
Output Data Set
Output Data Set
MDAT
MDATFLAG
MDATLCK
MDATLCKO
MDATRCT
MDATMDCB
MDATQDCB
MDATQDCB
SCVT
SCVTMWA
B
SCVTMSGH

Figure 2-59 (3 Of 4) - Message Handler Initialization

Figure 2-59 (4 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	When message output data sets DD names are specified in Array DOMXSMPC, DPPMINIT will do the following:		DPPMINIT
	A. Move all message output data sets DCBs to MDT.		
	B. Open message output data sets DCBs as QSAM output data sets. If any message output data set is not opened, error message DPP898 will be issued.	DPP898I	
	C. Place address of all message output data sets in MDAT.		DPPMINIT
2	The MESSAGE macro processor (DPPMMSG) will be loaded into core and its address will be placed in the SCVT.		DPPMINIT
3	The address of MDAT will be placed in the SCVT.		DPPMINIT
4	A Special Real Time Operating System lock control block will be built for DPPMMSG and it's address will be placed in MDAT.		DPPMINIT
5	A Special Real Time Operating System lock control block will be built for DPPMMSG1, and its address will be placed in MDAT.		DPPMINIT
6	The address of array DOMXSMRC will be placed in MDAT.		DPPMINIT



1 Save Address Of MESSAGE Macro Expansion.

2 Find Address Of The SCVT And From The SCVT Get The Address Of The MDAT And From The MDAT Get The Address Of The Lock Control Block For DPPMMSG. LOCK DPPMMSG.

3 Get Address Of Message DCB From MDAT.

4 Find Specified Message On Message Data Set.

If Message Not Found. Set Return Code To 8.

If Variable Count In Message Macro Expansion Less Than Defined (DEFMSG MACRO) Message Set Return Code To 4

Else If Variable Count In Message Macro Expansion Greater Than Defined Message. Set Return Code To 2.

5 Format Message With Time And/Or Date, Action Code And All User Variables.

6 If User Message Return Area Specified Pass Message To User.

7 Check Routing Codes For Validity.

If Routing Codes Not Valid Set Return Code To 12

Else PATCH DPPMMSG1 (Message Output Routine)

Figure 2-60 (1 Of 2) - Message Macro Processor

Figure 2-60 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	On entry to DPPMMSG, register 1 will contain the address of the MESSAGE macro expansion.		DPPMMSG
2	A Special Real Time Operating System LOCK macro will be issued for DPPMMSG with the lock control block found at location MDATLCK in the MDAT.		DPPMMSG
3	Save address of the message data set DCB found at location MDATMDCB in the MDAT.		DPPMMSG
4	If the specified message is found in the message data set, it will be read into virtual storage. If the message is not found, the return code will be set to 8.		DPPMMSG
5	The specified message will be formatted with all variables converted to EBCDIC. The time is converted to (HH:mm:SS.t, HH-Hours, MM-Minutes, SS-seconds, t-tenths of seconds), and if requested the date (DD/MMM/YY, DD-DAY, MMM-Month, YY-year) and the action code (I-information, A-action, D-decision).		DPPMMSG
6	If the address of a user return area is specified in the MESSAGE macro expansion, the message will be moved into the area.		DPPMMSG
7	If no routing codes are passed in the MESSAGE macro expansion, the one on the defined (DEFMSG) message will be used. The routing codes are checked against the valid routing codes in the DOMXSMPC array (RCT message routing code table) in the data base. If the routing codes are not valid, the return code is set to 12. If the routing codes are valid, the message will be passed to DPPMMSG1 via a PATCH macro. Routing code 255 is a no operation (DPPMMSG1 is not patched).		DPPMMSG

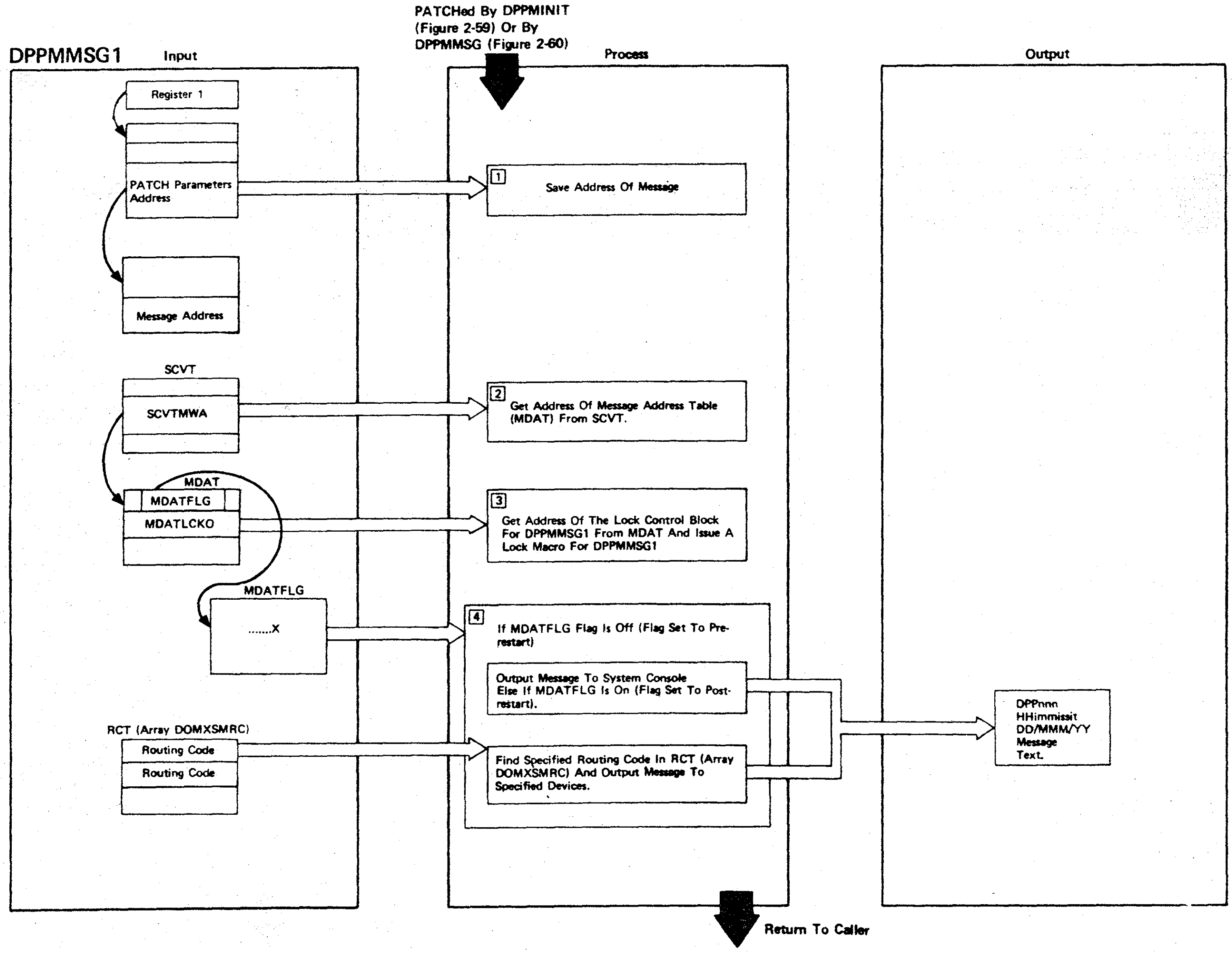

DPPMMSG1
Input
PATCHed By DPPMINIT (Figure 2-59) Or By DPPMMSG (Figure 2-60)
Process
Output
Register 1
PATCH Parameters Address
Message Address
SCVT
SCVTMWA
MDAT
MDATFLG
MDATLCKO
MDATFLG
.......X
RCT (Array DOMXSMRC)
Routing Code
Routing Code
1 Save Address Of Message
2 Get Address Of Message Address Table (MDAT) From SCVT.
3 Get Address Of The Lock Control Block For DPPMMSG1 From MDAT And Issue A Lock Macro For DPPMMSG1
4 If MDATFLG Flag Is Off (Flag Set To Pre-restart)
Output Message To System Console Else If MDATFLG Is On (Flag Set To Post-restart).
Find Specified Routing Code In RCT (Array DOMXSMRC) And Output Message To Specified Devices.
DPPnnn
HHimmissit
DD/MMM/YY
Message
Text.
Return To Caller

Figure 2-61 (1 Of 2) - Message Output Routine

Figure 2-61 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Save address of message.		DPPMMSG1
2	The MDAT is built by DPPMINIT and its address is stored in the SCVT at location SCVTMWA.		DPPMMSG1
3	The address of the Special Real Time Operating System Lock Control Block for DPPMMSG1 is found in the MDAT at location MDATLCKO. A LOCK macro using this lock control block will be issued to lock DPPMMSG1.		DPPMMSG1
4	If the MDATFLG flag is off (flag set to prerestart) and a restart is to be taken, issue all messages to the system console before the restart.		DPPMMSG1
	If the MDATFLG flag is on (flag set to post-restart), find the specified routing codes in RCT (Array DOMXSMRC). The formatted message is then output to the devices specified in the table.		

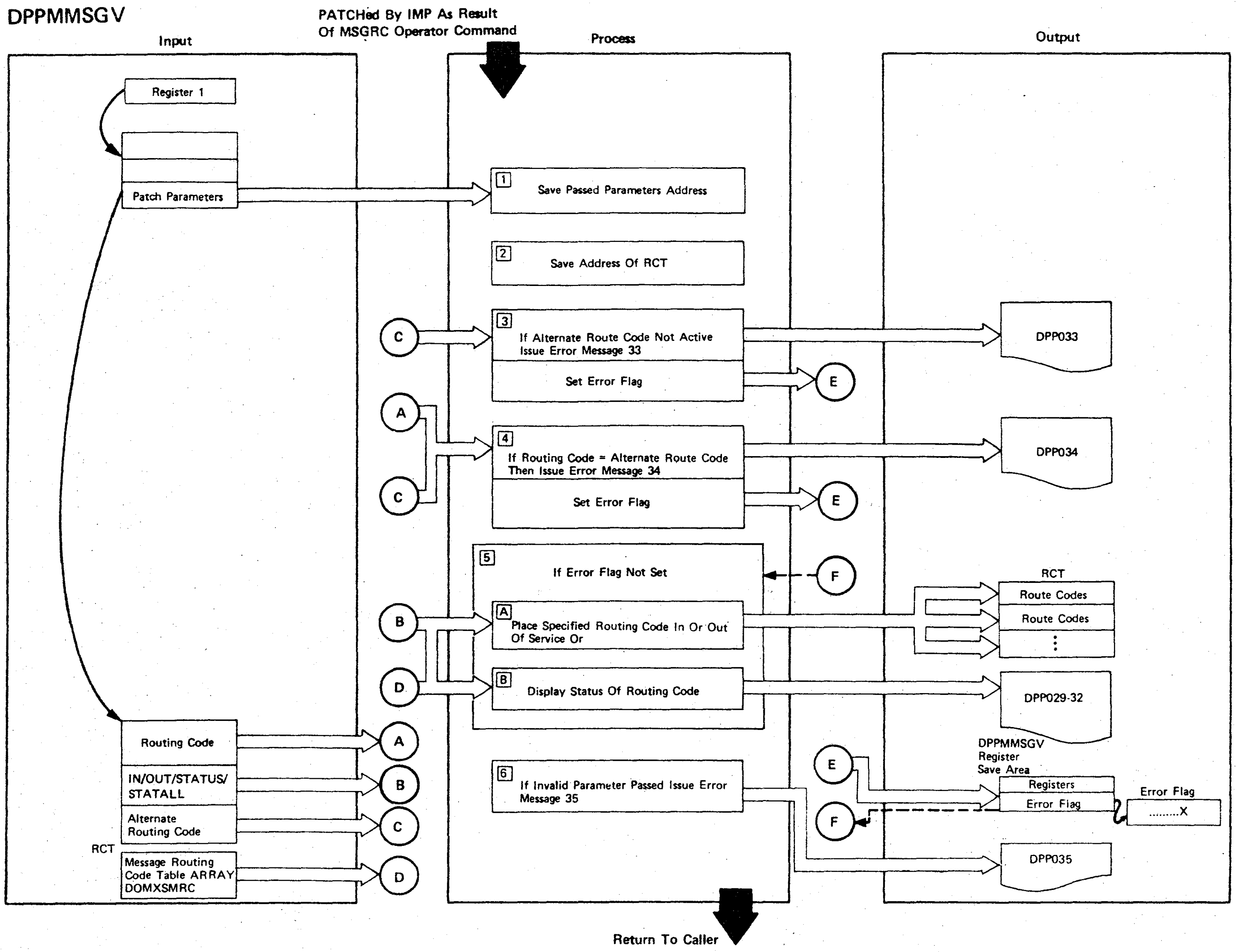


Figure 2-62 (1 Of 2) - Message Routing Code Status Change Facility

Figure 2-62 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Save the address of the passed parameters.		DPPMMSGV
2	Save the address of the RCT (message routing code table array DOMXSMRC).		DPPMMSGV
3	If an alternate routing code is passed which is not active, issue error message 33 and set the error flag in the register save area used by DPPMMSGV pointed to by register 13.	DPP033I	DPPMM8GV
4	If the routing code = alternate route code, issue error message 34 and set the error flag in the register save area used by DPPMMSGV pointed to by register 13.	DPP034I	DPPMMSGV
5	If the error flag is not on: Place the routing code in service if in-parameter passed. If the out-parameter passed, place routing code out-of-service.		
	If the STATUS or STATALL parameter passed, display status of routing code(s) via system messages 29 through 32.	DPP029I DPP030I DPP031I DPP032I	
6	If invalid parameters passed, issue error message 35.	DPP035I	

Input Message Processing

The Special Real Time Operating System provides a facility to allow for operator - Special Real Time Operating System communication or for the operator to communicate with a subsystem. This facility is the Input Message Processor. The Special Real Time Operating System, during initialization, issues a WTOR and leaves the reply outstanding. At a later time, the operator may reply with a predefined IMP command. The Input Message WTOR routine, DPPXIMPW, receives control and as the result of this reply patches the input message processing routine, DPPXIMPP. DPPXIMPP is responsible for validating the operator command and patching the specified user routine.

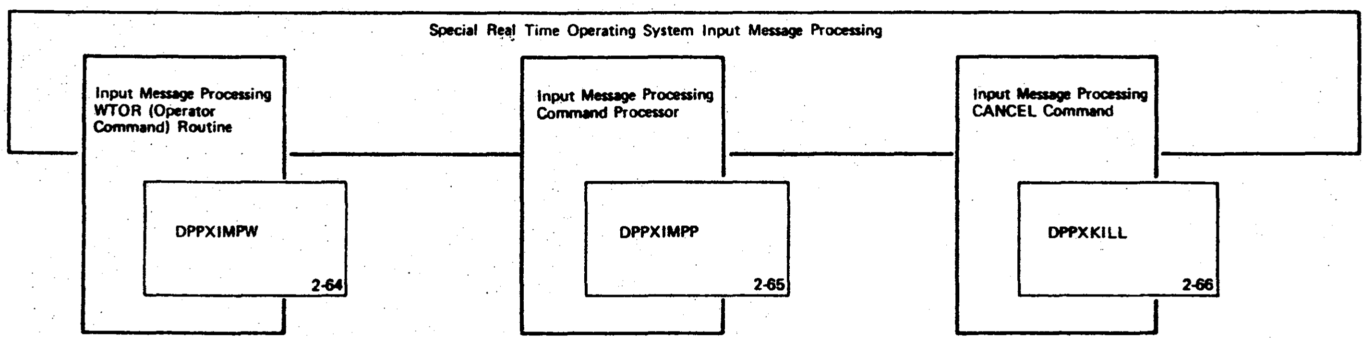

Special Real Time Operating System Input Message Processing
Input Message Processing WTOR (Operator Command) Routine
DPPXIMPW
2-64
Input Message Processing Command Processor
DPPXIMPP
2-65
Input Message Processing CANCEL Command
DPPXKILL
2-66

Figure 2-63 - Special Real Time Operating System Input Message Processing Overview

DPPXIMPW

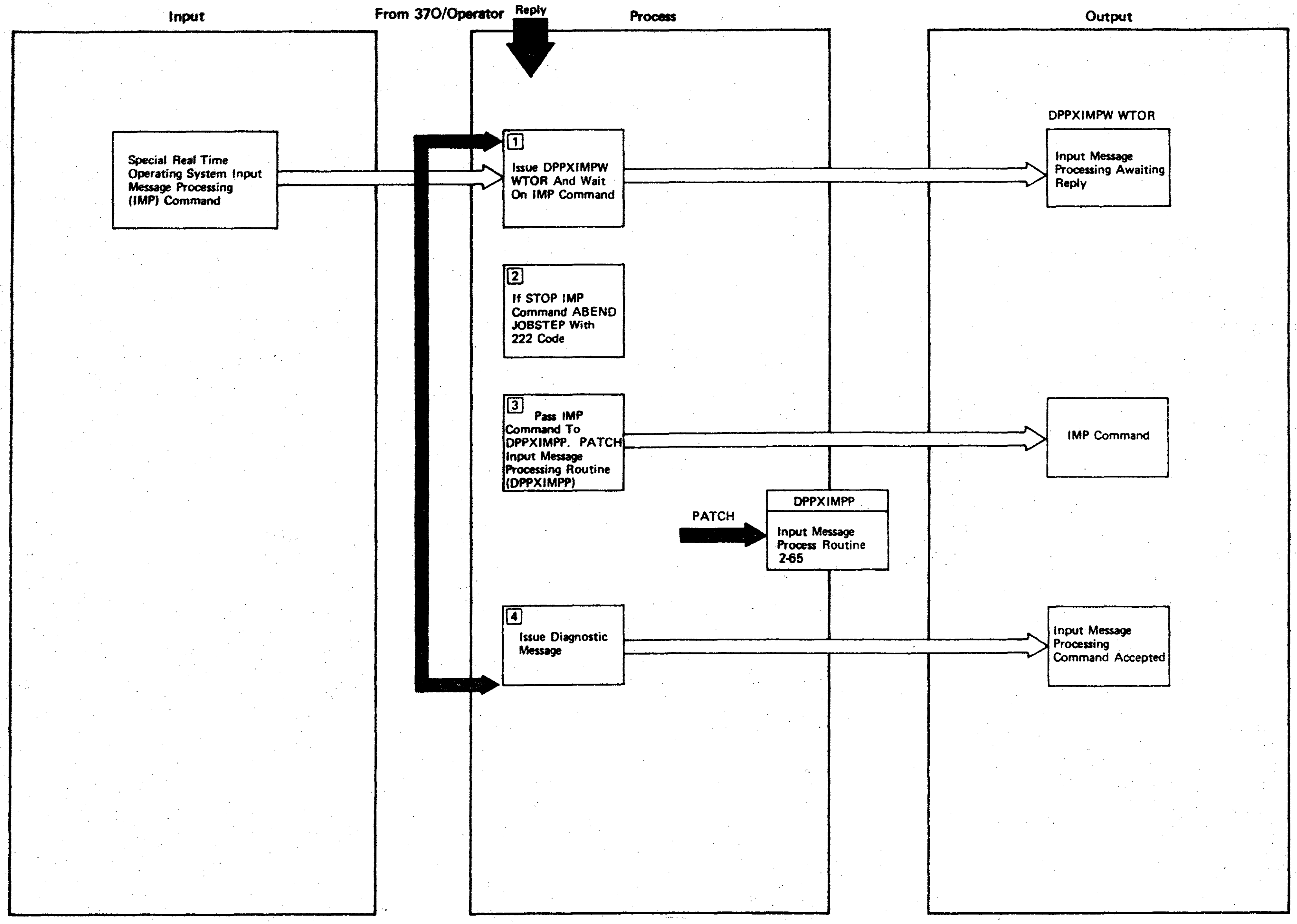

Input
From 370/Operator
Reply
Process
Output
Special Real Time Operating System Input Message Processing (IMP) Command
1
Issue DPPXIMPW WTOR And Wait On IMP Command
2
If STOP IMP Command ABEND JOBSTEP With 222 Code
3
Pass IMP Command To DPPXIMPP. PATCH Input Message Processing Routine (DPPXIMPP)
PATCH
DPPXIMPP
Input Message Process Routine 2-65
4
Issue Diagnostic Message
DPPXIMPW WTOR
Input Message Processing Awaiting Reply
IMP Command
Input Message Processing Command Accepted

Figure 2-64 (1 Of 2) - Input Message WTOR Routine

Figure 2-64 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The System/370 operator will issue an IMP Command to a WTOR issued by DPPXIMPW. INPUT MESSAGE PROCESSING AWAITING REPLY.		DPPXIMPW
2	If the STOP command is entered, ABEND job step with dump code 222.	USER 222	DPPXIMPW
3	The IMP command will be passed to DPPXIMPP via PATCH.		DPPXIMPW
4	DPPXIMPW will issue a diagnostic WTO message to the System/370 operator. INPUT MESSAGE PROCESSING COMMAND ACCEPTED.		DPPXIMPW

DPPXIMPP

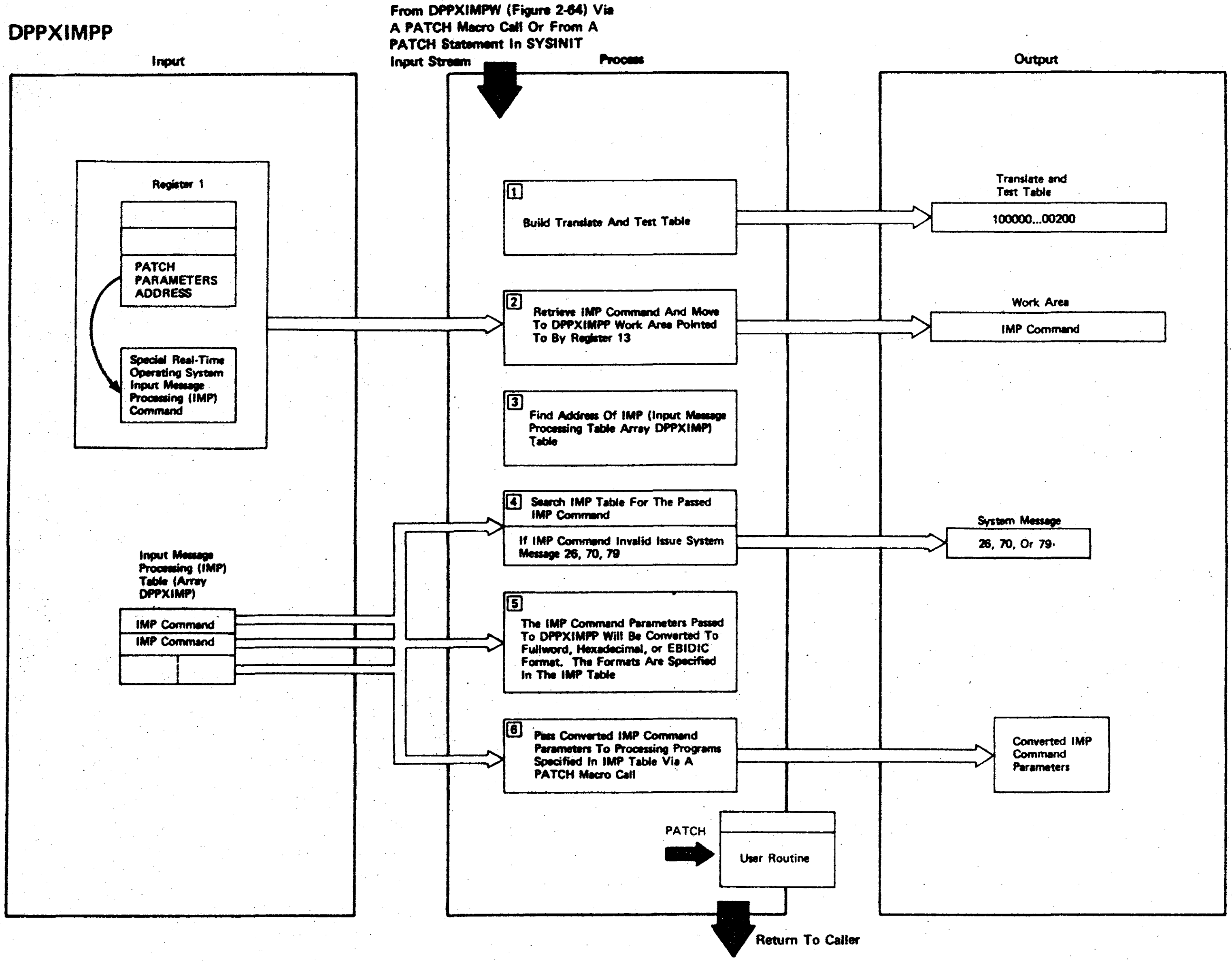

From DPPXIMPW (Figure 2-64) Via A PATCH Macro Call Or From A PATCH Statement In SYSINIT Input Stream
Input
Process
Output
Register 1
PATCH PARAMETERS ADDRESS
Special Real-Time Operating System Input Message Processing (IMP) Command
Input Message Processing (IMP) Table (Array DPPXIMP)
IMP Command
IMP Command
1 Build Translate And Test Table
2 Retrieve IMP Command And Move To DPPXIMPP Work Area Pointed To By Register 13
3 Find Address Of IMP (Input Message Processing Table Array DPPXIMP) Table
4 Search IMP Table For The Passed IMP Command
If IMP Command Invalid Issue System Message 26, 70, 79
5 The IMP Command Parameters Passed To DPPXIMPP Will Be Converted To Fullword, Hexadecimal, or EBIDIC Format. The Formats Are Specified In The IMP Table
6 Pass Converted IMP Command Parameters To Processing Programs Specified In IMP Table Via A PATCH Macro Call
PATCH
User Routine
Return To Caller
Translate and Test Table
100000...00200
Work Area
IMP Command
System Message
26, 70, Or 79.
Converted IMP Command Parameters

Figure 2-65 (1 Of 2) - Input Message Processing Routine

Figure 2-65 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	A 255 byte translate and test table will be built where byte 0 is 1 and byte 107 is 2 and the other 253 bytes are set to 0.		DPPXIMPP
2	Register 13 points to a work area used by DPPXIMPP. The IMP command will be moved to this area.		DPPXIMPP
3	The address of the IMP table (array DPPXIMP) will be obtained via a GETARRAY nacro.		DPPXIMPP
4	A search is made of the IMP table for the passed IMP command. If the command is invalid, system message 26, 70, 72, or 79 will be issued.	DPP026I DPP070I DPP072I DPP079I	DPPXIMPP
5	The IMP table contains the parameter format for the passed IMP Command parameters. The parameters will be converted to fullword, hexadecimal, or EBCDIC format.		DPPXIMPP
6	The IMP table contains the name of the processing programs for all IMP commands. The IMP command converted parameters will be passed to the processing program via a PATCH macro to either the MASTER or SLAVE partition.		DPPXIMPP

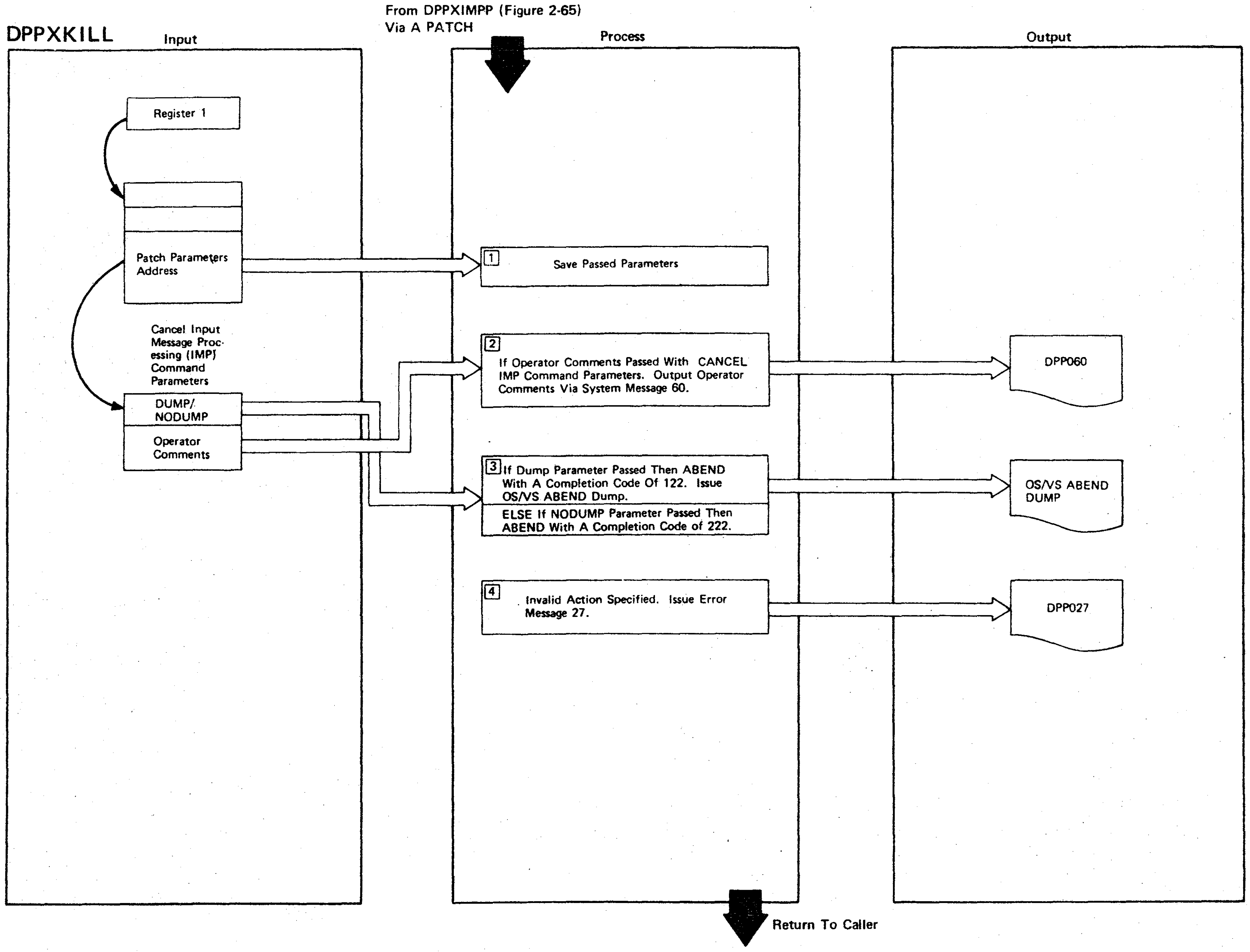

DPPXKILL
Input
From DPPXIMPP (Figure 2-65)
Via A PATCH
Process
Output
Register 1
Patch Parameters Address
Cancel Input Message Processing (IMP) Command Parameters
DUMP/ NODUMP
Operator Comments
1 Save Passed Parameters
2 If Operator Comments Passed With CANCEL IMP Command Parameters. Output Operator Comments Via System Message 60.
3 If Dump Parameter Passed Then ABEND With A Completion Code Of 122. Issue OS/VS ABEND Dump.
ELSE If NODUMP Parameter Passed Then ABEND With A Completion Code of 222.
4 Invalid Action Specified. Issue Error Message 27.
DPP060
OS/VS ABEND DUMP
DPP027
Return To Caller

Figure 2-66 (1 Of 2) - Cancel Routine

Figure 2-66 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Save address of the CANCEL Input Message Processing (IMP) command parameters.		DPPXKILL
2	If operator comments passed with the CANCEL IMP parameter, output the operator comments via message 60.	DPP060I	DPPXKILL
3	If the DUMP parameter is passed, issue ABEND macro with the dump option and a completion code of 122.	USER 122	DPPXKILL
	If the NODUMP parameter is passed, issue ABEND macro without the dump option and a completion code of 222.	USER 222	
4	If action requested is not DUMP or NODUMP, an error message is issued.	DPP027I	DPPXKILL

Report Data Output

The report data output facility provides the capability of transferring user-generated data from one or more user-defined sequential data sets to a single user-defined sequential data set. The report data output facility is invoked through a REPORT input message processing command.

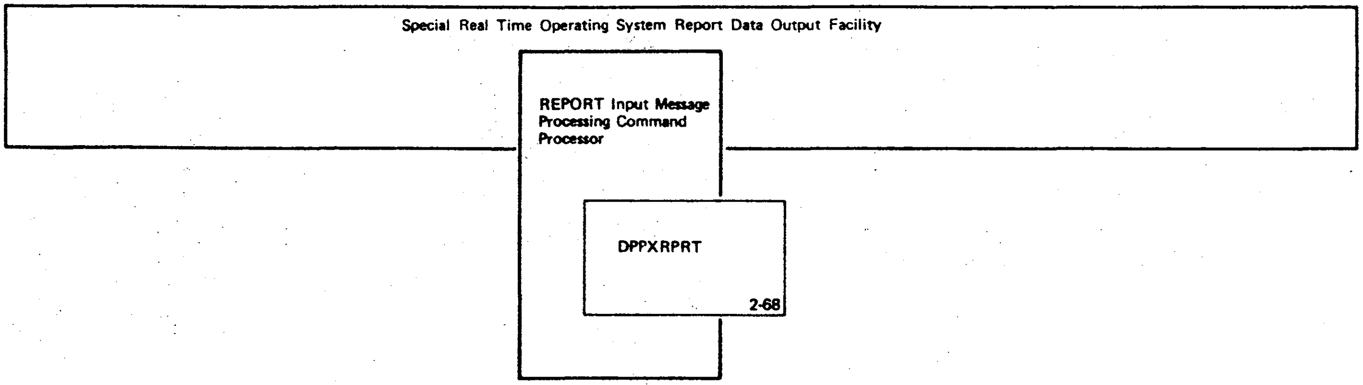

Special Real Time Operating System Report Data Output Facility
REPORT Input Message Processing Command Processor
DPPXRPRT
2-68

Figure 2-67 Special Real Time Operating System Report Data Output Facility Overview

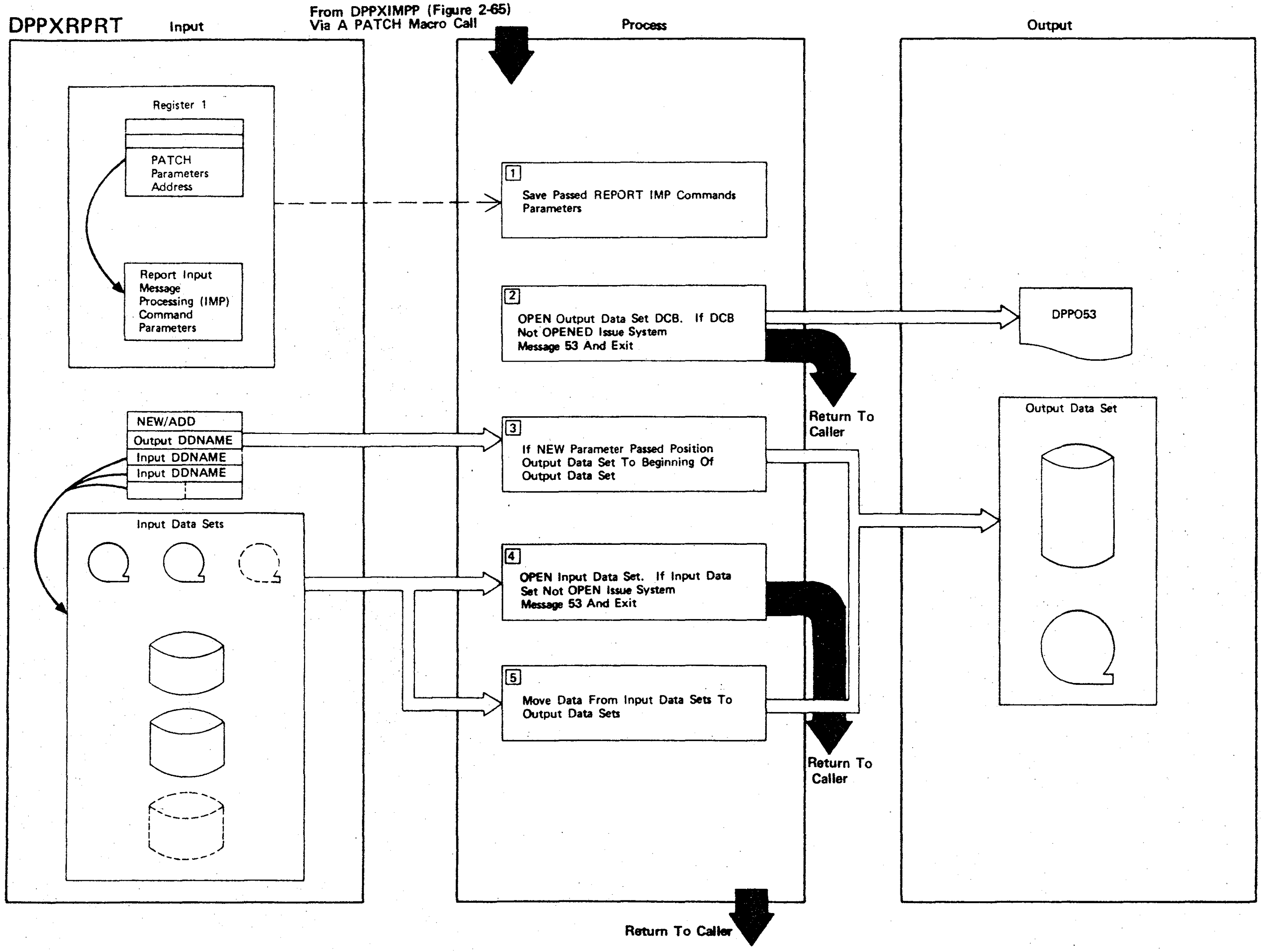

DPPXRPRT
Input
From DPPXIMPP (Figure 2-65)
Via A PATCH Macro Call
Process
Output
Register 1
PATCH Parameters Address
Report Input Message Processing (IMP) Command Parameters
1 Save Passed REPORT IMP Commands Parameters
2 OPEN Output Data Set DCB. If DCB Not OPENED Issue System Message 53 And Exit
DPPO53
Return To Caller
NEW/ADD
Output DDNAME
Input DDNAME
Input DDNAME
3 If NEW Parameter Passed Position Output Data Set To Beginning Of Output Data Set
Output Data Set
Input Data Sets
4 OPEN Input Data Set. If Input Data Set Not OPEN Issue System Message 53 And Exit
5 Move Data From Input Data Sets To Output Data Sets
Return To Caller
Return To Caller

Figure 2-68 (1 Of 2) - Report Data Output Facility

Figure 2-68 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Save the address of the passed REPORT IMP command parameters.		DPPXRPRT
2	OPEN the passed output data set. If the data set was not opened, issue system message 53.	DPP053I	
3	If the NEW parameter was passed with the REPORT IMP command, the output data set will be positioned at the beginning of the output data set.		DPPXRPRT
4	Open the passed input data set. If the data set was not opened, issue system message 53.	DPP053I	DPPXRPRT
5	All data in the input data sets is moved into the output data set. The output data set must be large enough to contain the data.		DPPXRPRT

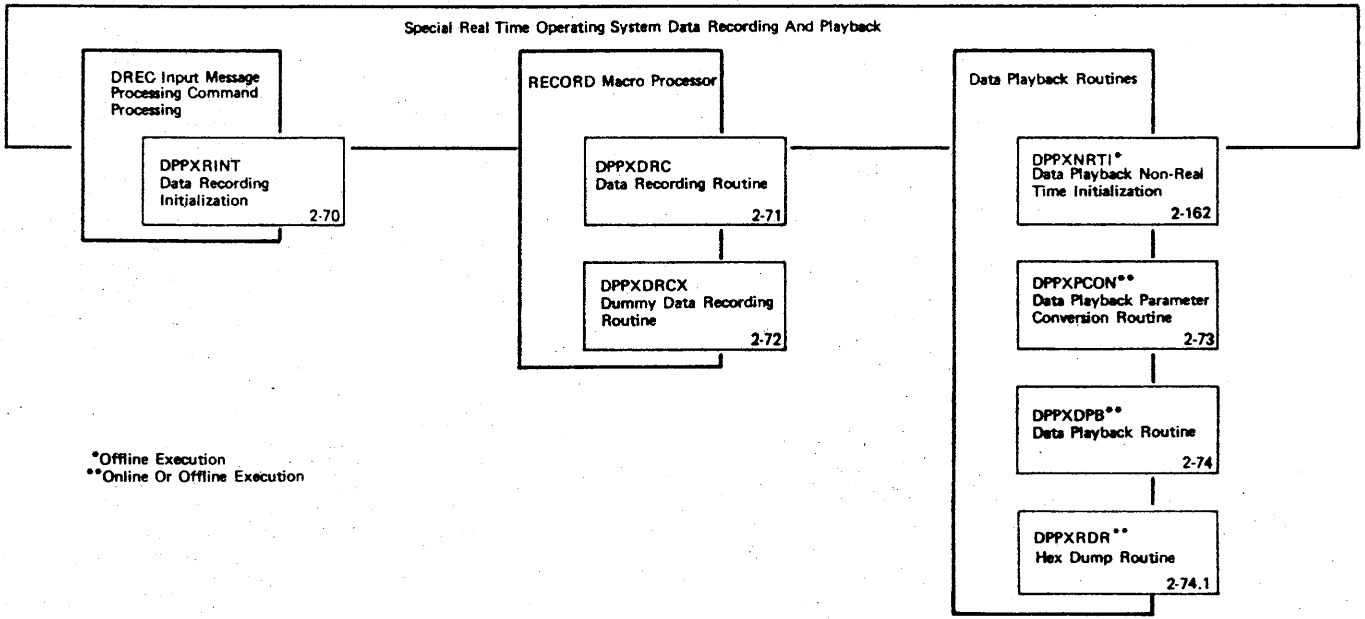
Data Recording and Playback

Data recording and playback enable the user to record data in a realtime environment and to play it back at a later time either in a realtime environment or in an offline environment.

Data recording is enabled (or disabled) by DPPXRINT as the result of a DREC operator command. When data recording is enabled DPPXDRC receives control from the user via a RECORD macro call and is responsible for formatting and recording the requested data. When data recording is disabled a stub routine, DPPXDRCX, replaces DPPXDRC. DPPXDRCX sets a return code and returns to the user without recording any data.

Data playback is initiated in the online system by a PATCH to module DPPXPCON. This routine is responsible for converting the input playback parameters to a form acceptable to the playback routine, DPPXDPB. DPPXDPB gains control from DPPXPCON by a LINK macro call and is responsible for playing back the requested data.

Data playback is initiated in the offline system by executing module DPPXNRTI on an EXEC statement specifying PGM=DPPXNRTI. This routine is responsible for building a parameter list in a form acceptable to DPPXPCON and then linking to that routine. Once DPPXPCON receives control, the playback operation is the same as for the online system previously described.


Special Real Time Operating System Data Recording And Playback
DREC Input Message Processing Command Processing
DPPXRINT
Data Recording Initialization
2-70
RECORD Macro Processor
DPPXDRC
Data Recording Routine
2-71
DPPXDRCX
Dummy Data Recording Routine
2-72
Data Playback Routines
DPPXNRTI*
Data Playback Non-Real Time Initialization
2-162
DPPXPCON**
Data Playback Parameter Conversion Routine
2-73
DPPXDPB**
Data Playback Routine
2-74
DPPXRDR**
Hex Dump Routine
2-74.1
*Offline Execution
**Online Or Offline Execution

69 - Special Real Time Operating System Data Record And Playback Overview

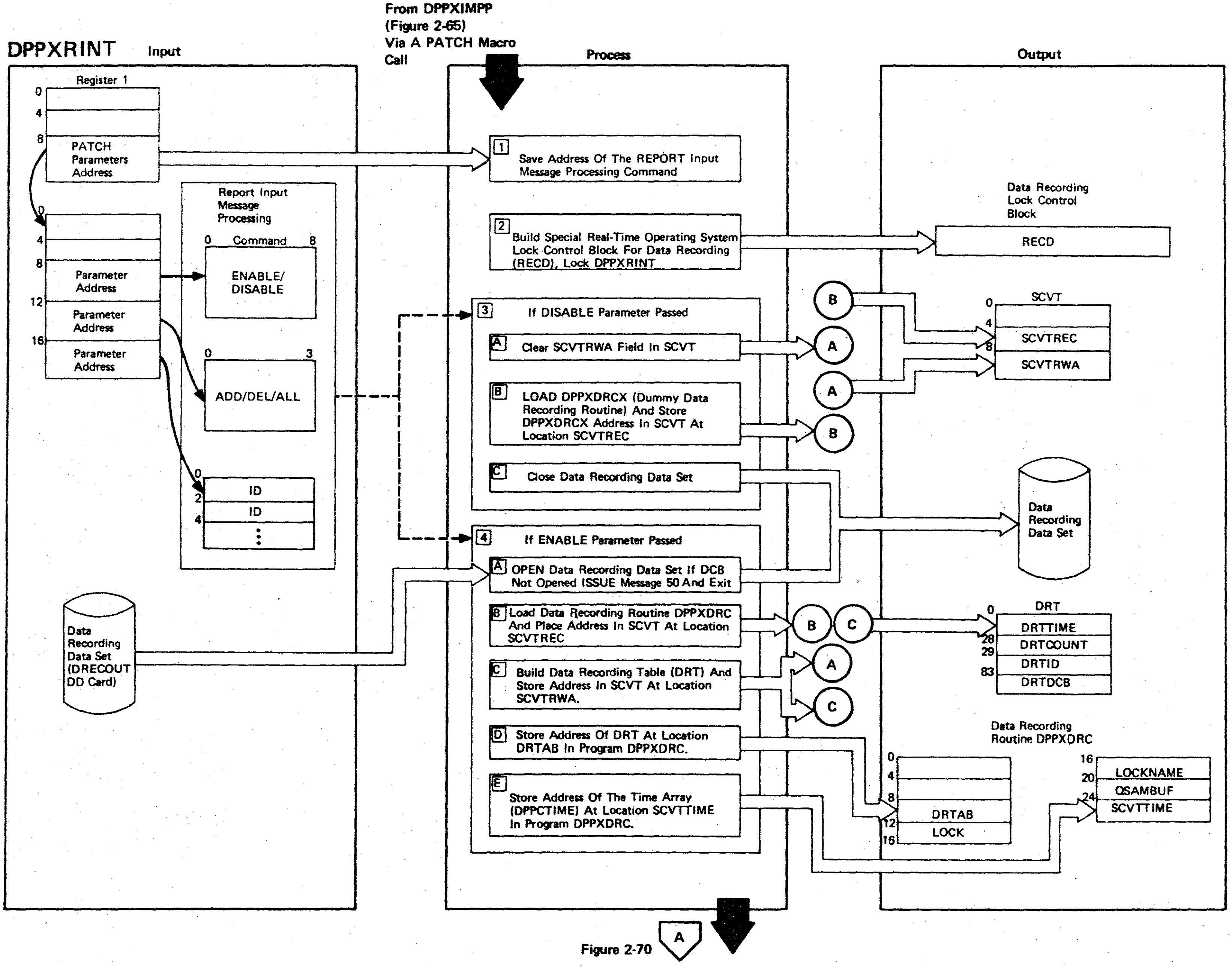
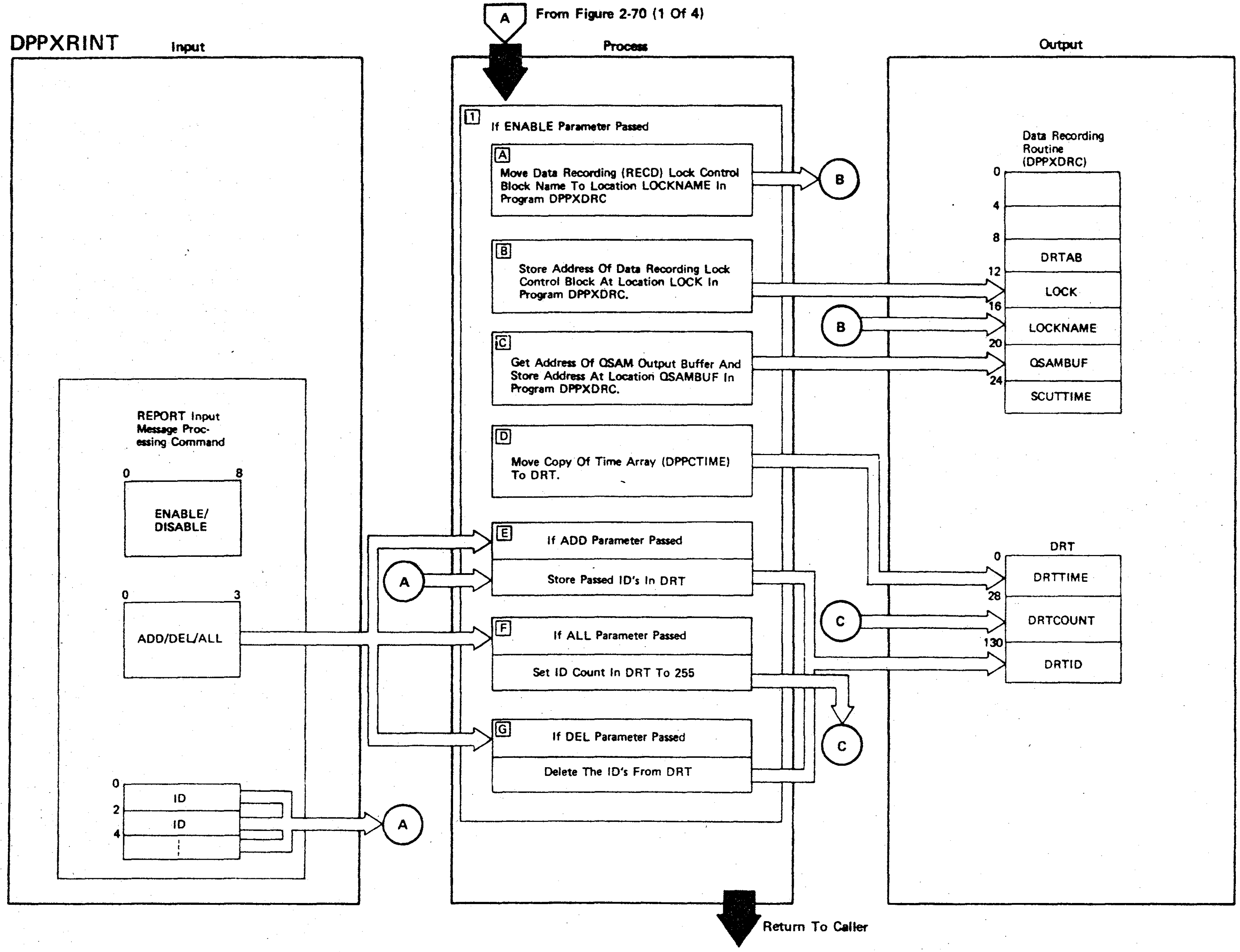

DPPXRINT
Input
From DPPXIMPP (Figure 2-65) Via A PATCH Macro Call
Process
Output
Register 1
0
4
8
PATCH Parameters Address
0
4
8
Parameter Address
12
Parameter Address
16
Parameter Address
Report Input Message Processing
0 Command 8
ENABLE/ DISABLE
0 3
ADD/DEL/ALL
0
ID
2
ID
4
Data Recording Data Set (DRECOUT DD Card)
1 Save Address Of The REPORT Input Message Processing Command
2 Build Special Real-Time Operating System Lock Control Block For Data Recording (RECD), Lock DPPXRINT
3 If DISABLE Parameter Passed
A Clear SCVTRWA Field In SCVT
B LOAD DPPXDRCX (Dummy Data Recording Routine) And Store DPPXDRCX Address In SCVT At Location SCVTREC
C Close Data Recording Data Set
4 If ENABLE Parameter Passed
A OPEN Data Recording Data Set If DCB Not Opened ISSUE Message 50 And Exit
B Load Data Recording Routine DPPXDRC And Place Address In SCVT At Location SCVTREC
C Build Data Recording Table (DRT) And Store Address In SCVT At Location SCVTRWA.
D Store Address Of DRT At Location DRTAB In Program DPPXDRC.
E Store Address Of The Time Array (DPPCTIME) At Location SCVTTIME In Program DPPXDRC.
Data Recording Lock Control Block
RECD
B
A
A
B
SCVT
0
4
SCVTREC
8
SCVTRWA
Data Recording Data Set
B
C
A
C
DRT
0
DRTTIME
28
DRTCOUNT
29
DRTID
83
DRTDCB
Data Recording Routine DPPXDRC
0
4
8
DRTAB
12
LOCK
16
16
LOCKNAME
20
QSAMBUF
24
SCVTTIME
Figure 2-70
A

Figure 2-70 (1 Of 4) - Data Recording Initialization Routine

Figure 2-70 (2 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	When DPPXRINT is entered, the address of the REPORT input message processing command will be saved.		DPPXRINT
2	A Special Real Time Operating System DEFLOCK macro will be issued to build a LOCK control block (RECD) for data recording routines (DPPXRINT and DPPXDRC). A LOCK macro will be issued to lock DPPXRINT.		DPPXRINT
3	If the DISABLE parameter was passed. A. Clear location SCVTRWA in the SCVT. B. Issue a LOAD macro for DPPXDRCX (dummy data recording routine) and the address of DPPXDRCX will be stored at location SCVTREC in the SCVT. C. Close the data recording data set.		DPPXRINT
4	If ENABLE parameter was passed. A. OPEN data recording data set. If DCB is not opened, issue message 50. B. Issue a LOAD macro for DPPXDRC (data recording routine). C. Build data recording table (DRT) and store address at location SCVTRWA in the SCVT. D. Store address of DRT at location DRTAB in program DPPXDRC (the first 36 bytes of DPPXDRC are used as a control table). E. The Special Real Time Operating System time array (DPPCTIME) address will be stored at location SCVTTIME in program DPPXDRC (see 4-D).	DPP050I	DPPXRINT



1 If ENABLE Parameter Passed

A Move Data Recording (RECD) Lock Control Block Name To Location LOCKNAME In Program DPPXDRC

B Store Address Of Data Recording Lock Control Block At Location LOCK In Program DPPXDRC.

C Get Address Of QSAM Output Buffer And Store Address At Location QSAMBUF In Program DPPXDRC.

D Move Copy Of Time Array (DPPCTIME) To DRT.

E If ADD Parameter Passed

Store Passed ID's In DRT

F If ALL Parameter Passed

Set ID Count In DRT To 255

G If DEL Parameter Passed

Delete The ID's From DRT

Figure 2-70 (3 Of 4) - Data Recording Initialization Routine

Figure 2-70 (4 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the ENABLE parameter passed: A. Move data recording (RECD) lock control block name to location LOCKNAME in program DPPXDRC (see Figure 2-70 (2 of 4) 4-D). B. Store data recording (RECD) lock control block address at location lock in program DPPXDRC (see Figure 2-70 (2 of 4) 4-D). C. A GET macro will be issued for the address of an output buffer. The address of the output buffer will be stored at location QSAMBUF in program DPPXDRC (see Figure 2-70 (2 of 4) 4-D). D. A copy of the time array (DPPCTIME) will be moved to location DRTTIME in the DRT. E. If the ADD parameter passed, all passed IDs will be moved to the DRT at location DRTID. F. If the ALL parameter passed, the ID count (DRTCOUNT) in the DRT will be set to 255. G. If the DEL parameter passed, all passed IDs will be deleted from the DRT.		DPPXRINT

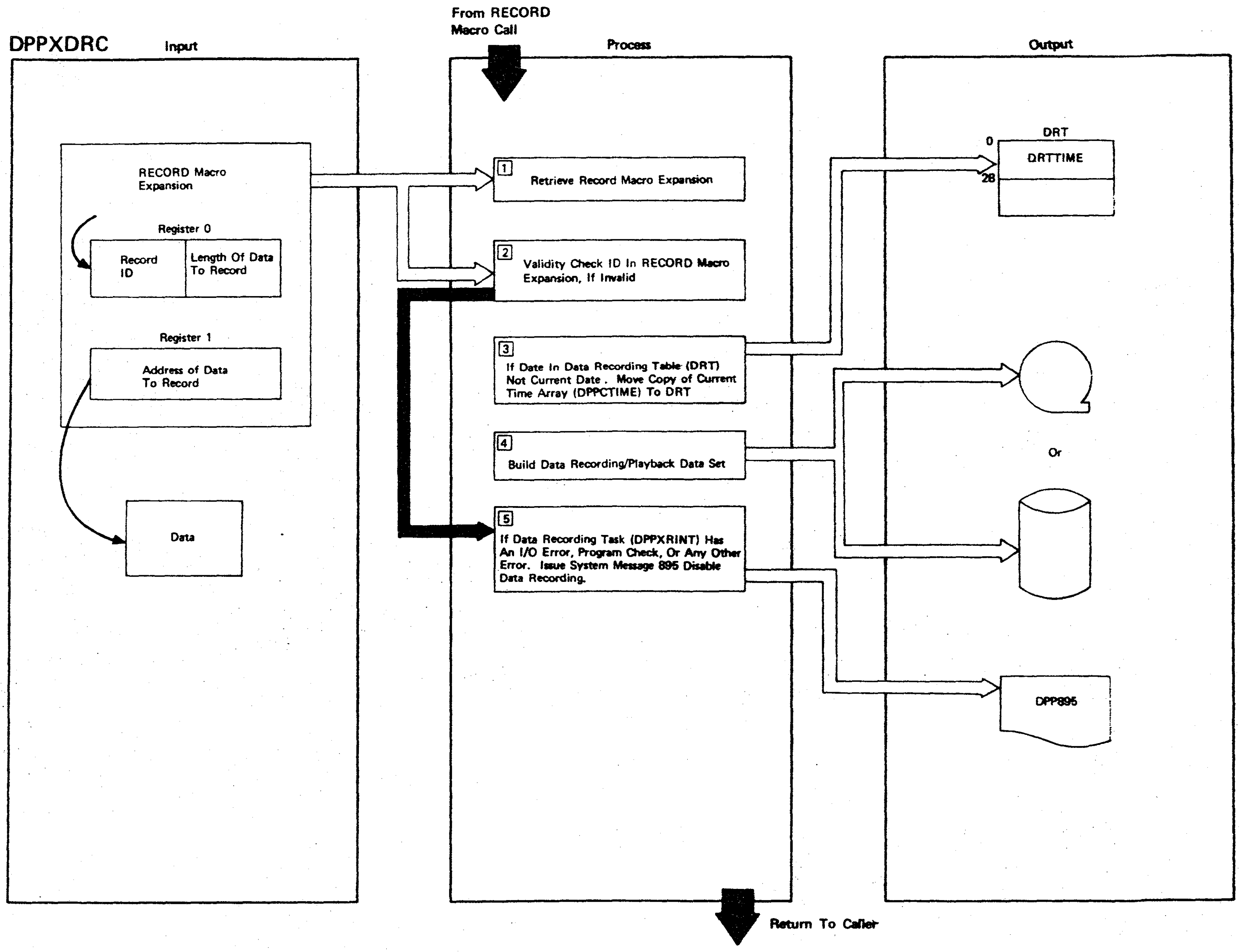

DPPXDRC
Input
From RECORD Macro Call
Process
Output
RECORD Macro Expansion
Register 0
Record ID
Length Of Data To Record
Register 1
Address of Data To Record
Data
1 Retrieve Record Macro Expansion
2 Validity Check ID In RECORD Macro Expansion, If Invalid
3 If Date In Data Recording Table (DRT) Not Current Date . Move Copy of Current Time Array (DPPCTIME) To DRT
4 Build Data Recording/Playback Data Set
5 If Data Recording Task (DPPXRINT) Has An I/O Error, Program Check, Or Any Other Error. Issue System Message 895 Disable Data Recording.
DRT
0
DRTTIME
28
Or
DPP895
Return To Caller

Figure 2-71 (1 Of 2) - Data Recording Routine

Figure 2-71 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	When DPPXDRC is entered, register 0 will contain in bits 0-15 the ID to be affixed to the data and in bits 16-31 the length of the data. Register 1 will contain the address of the data to record.		DPPXDRC
2	Contained in the Data Recording Table (DRT) are the IDs that may be used for data recording. The ID passed to data recording will be checked against this table. If the passed ID is in DRT or if the enable all flag is on, the data will be recorded, otherwise the return code is set to 8, and no further processing is performed.		DPPXDRC
3	When the time in the DRT is not current time, DPPXDRC will replace the time array contained in DRT with an updated copy.		DPPXDRC
4	The Data Recording/Playback data set built by DPPXDRC is a sequential QSAM data set that may be on a tape or a disk volume. The data set consists of three types of records: date records, pad records, and user records. A date record consists of the time array. The date record is written each time the DRT is updated. A pad record is written whenever the QSAM output buffer, used by DPPXDRC, contains less than 50 but more than zero bytes of data. A user record consists of the data that the user requested to be recorded.		DPPXDRC
5	If there is any error in the data recording task (DPPXRINT), message DPP895 will be issued and data recording will be disabled.	DPP895I	DPPXDRC

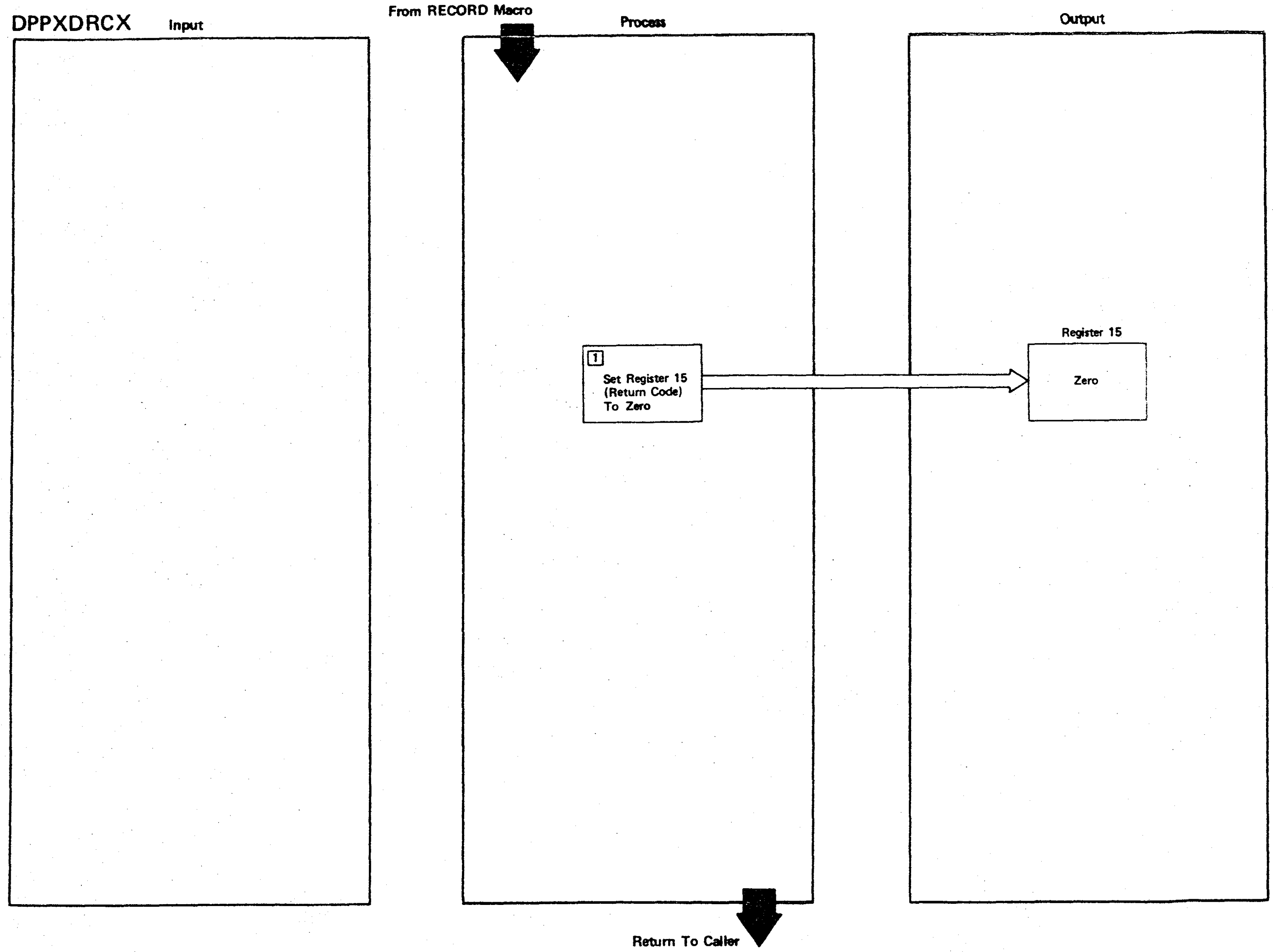


Figure 2-72 (1 Of 2) - Dummy Data Recording Routine

Figure 2-72 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	DPPXDRCX is a dummy data recording routine that sets register 15 to return code 4. The RECORD macro will branch to DPPXDRCX until the actual data recording routine (DPPXDRC) is initialized through the DREC IMP command.		DPPXDRCX

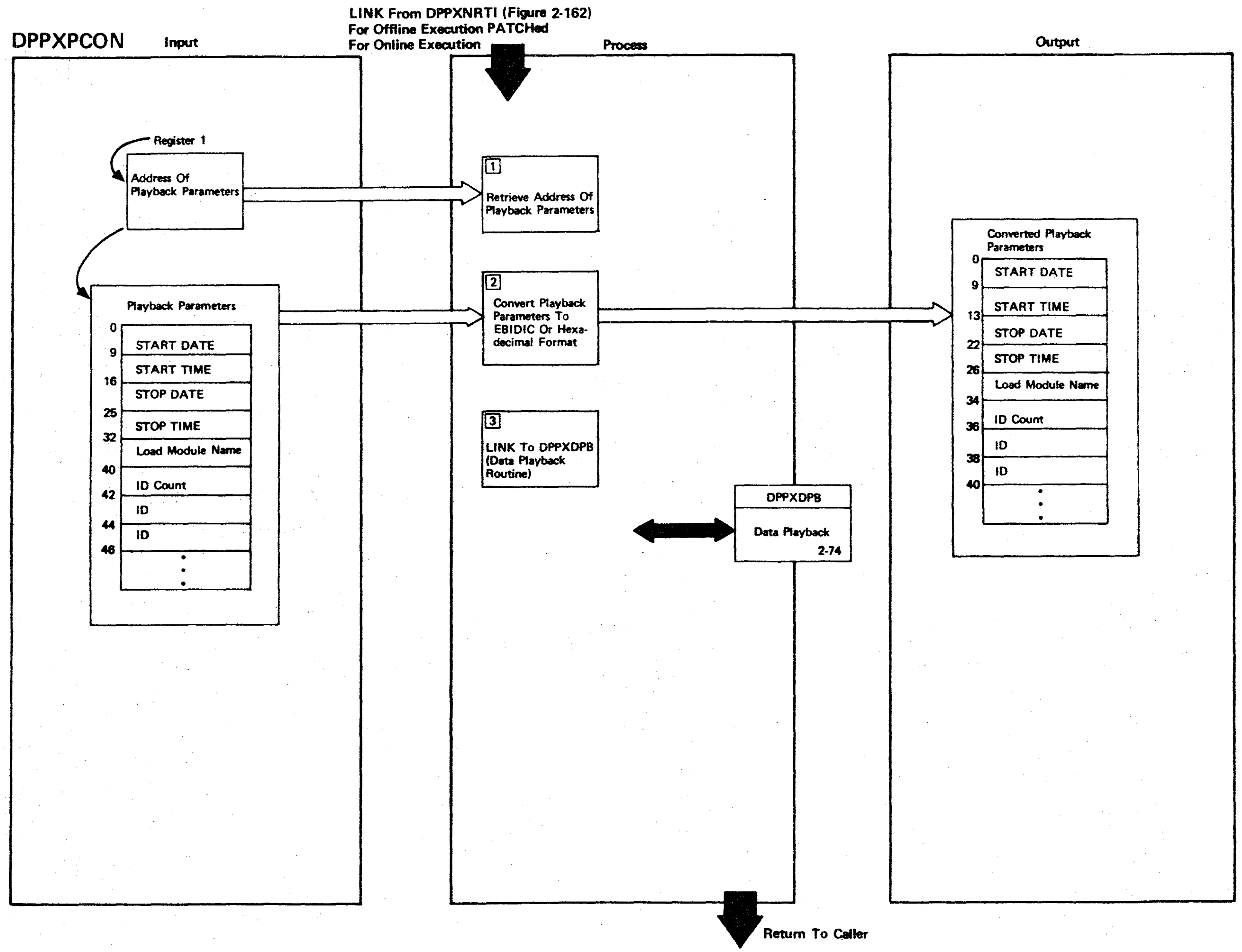


Figure 2-73 (1 Of 2) - Data Playback Parameter Conversion Routine

Figure 2-73 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	When DPPXPCON is entered, register 1 will contain the address of the playback parameters to process.		DPPXPCON
2	The playback parameters pointed to by register 1 will be moved to a work area for processing.		DPPXPCON
3	The start and stop dates and LOAD module will be converted to EBCDIC. The start and stop times will be converted to decimal data, and the ID count and IDs converted to hexadecimal data.		DPPXPCON
4	The converted playback parameters will be passed to DPPXDPB via a link.		DPPXPCON

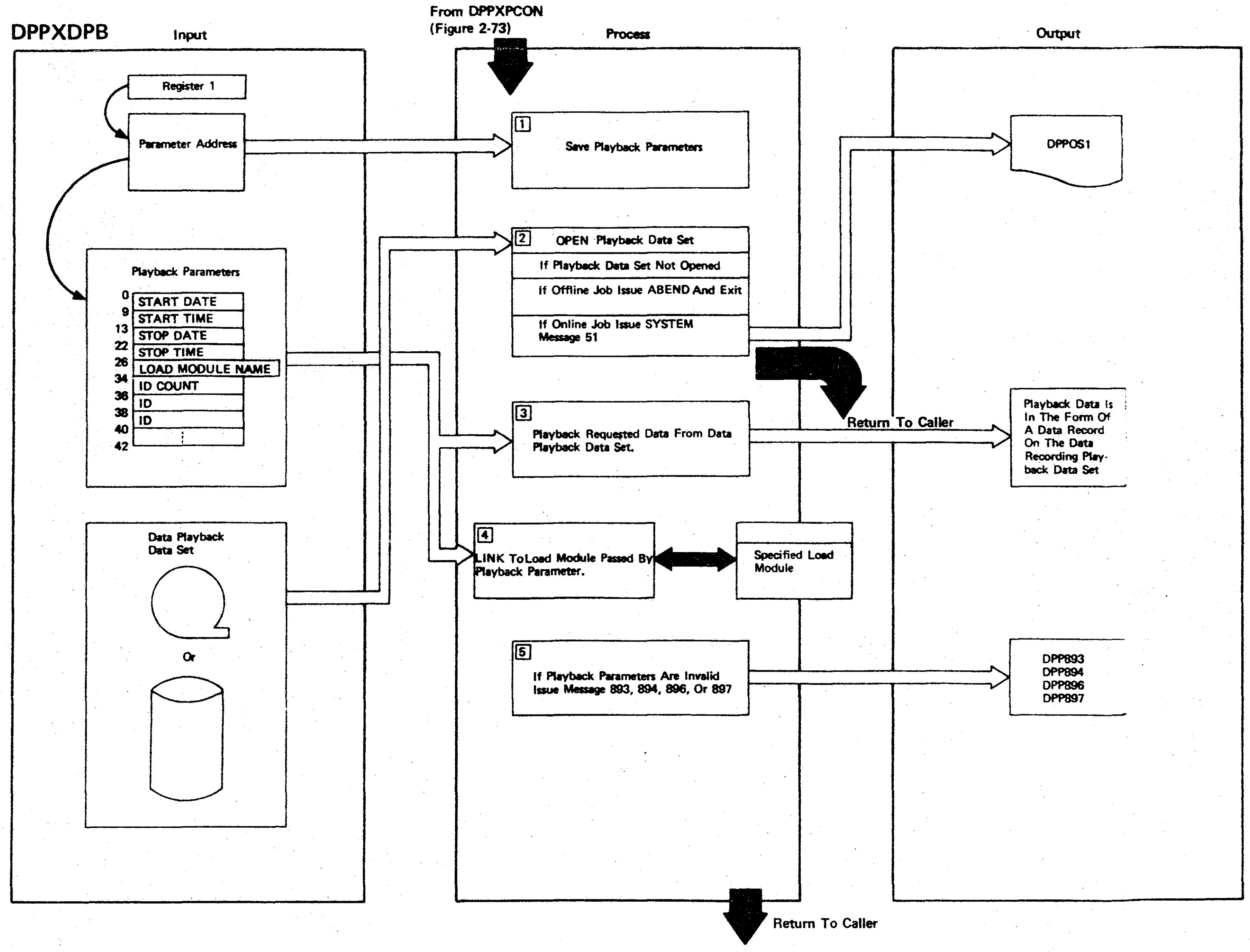

DPPXDPB
Input
From DPPXPCON
(Figure 2-73)
Process
Output
Register 1
Parameter Address
Playback Parameters
0 START DATE
9 START TIME
13 STOP DATE
22 STOP TIME
26 LOAD MODULE NAME
34 ID COUNT
36 ID
38 ID
40
42
Data Playback
Data Set
Or
1 Save Playback Parameters
2 OPEN Playback Data Set
If Playback Data Set Not Opened
If Offline Job Issue ABEND And Exit
If Online Job Issue SYSTEM Message 51
Return To Caller
3 Playback Requested Data From Data Playback Data Set.
4 LINK ToLoad Module Passed By Playback Parameter.
Specified Load Module
5 If Playback Parameters Are Invalid Issue Message 893, 894, 896, Or 897
DPPOS1
Playback Data Is In The Form Of A Data Record On The Data Recording Play-back Data Set
DPP893
DPP894
DPP896
DPP897
Return To Caller

Figure 2-74 (1 Of 2) - Data Playback Routine

Figure 2-74 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	When DPPXDPB is entered, register 1 will contain the address of the playback parameter.		DPPXDPB
2	The data recording/playback data set is a sequential QSAM data set built by DPPXDRC. The data set can be a tape or disk data set.		DPPXDPB
3	The data recording/playback data set is opened for input. If the data set is not opened and DPPXDRC is running offline (non-Special Real Time Operating System), DPPXDRC will ABEND. If DPPXDRC is running under Special Real Time Operating System, and the data set is not opened, DPPXDRC will issue message 51. In either case, no further processing is performed.	DPP051I USER 71	DPPXDPB
4	When the data was recorded, it was time tagged (date and time of day was added). In the playback parameters a time and ID range is specified. All data on the data recording/playback data set that falls within the specified time and ID range will be read in.		DPPXDPB
5	The data played back will be passed to a routine for processing. The data will be passed to a user routine if one is specified. If no user routine is specified, the data is passed to Special Real Time Operating System hexadecimal dump routine (DPPXRDR).	DPP893I DPP894I DPP896I DPP897I	DPPXDP8

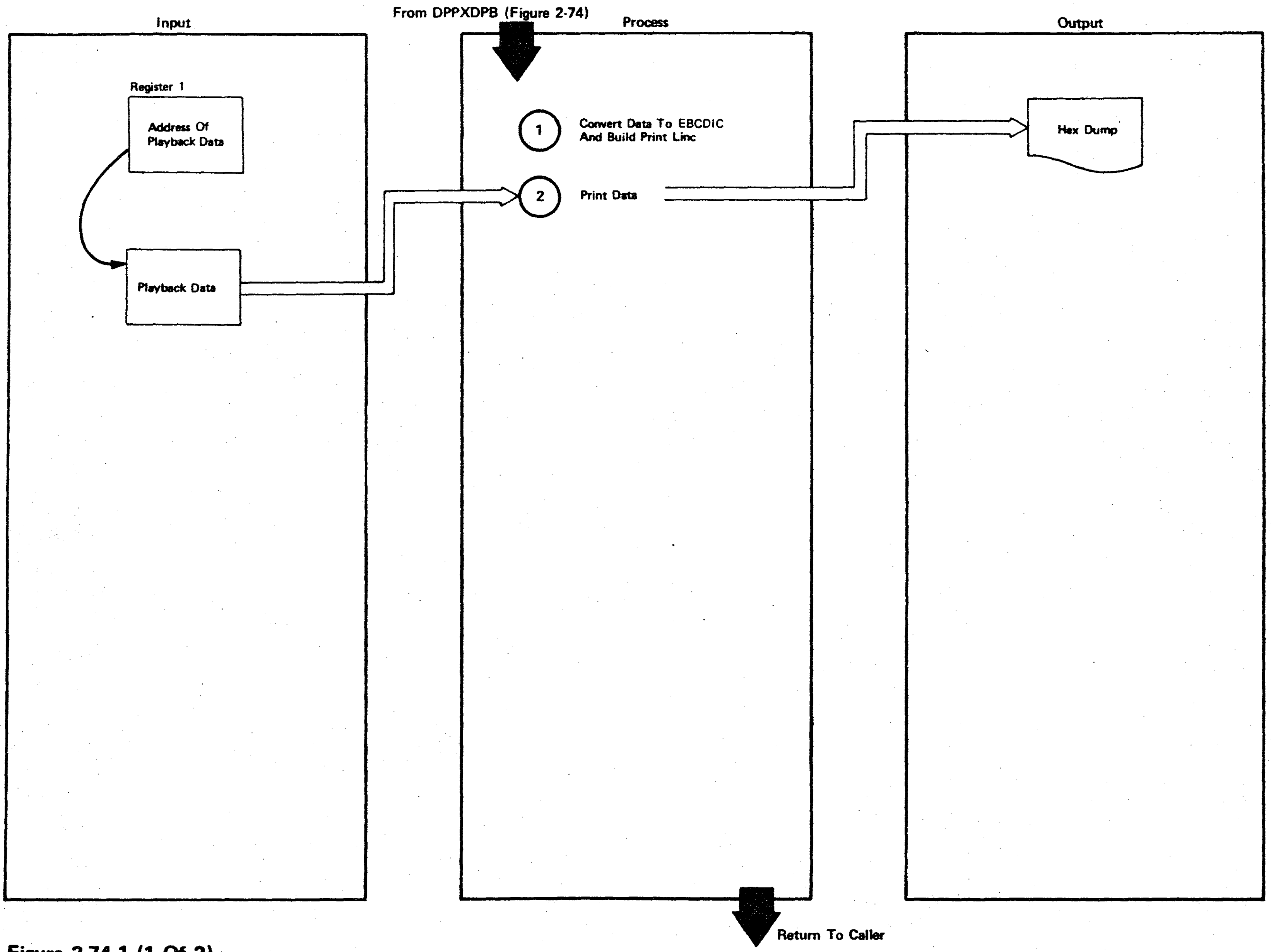


Figure 2-74.1 (1 Of 2)

Figure 2-74.1 (2 Of 2)

Step	Extended Description	Messages And ABEND Codes	PDL Segment
1	The Data Is Converted From Hexadecimal To EBIDIC For Printing. A Print Line Will Be Built In The Same Format As An OS/VS ABEND Dump Line.		DPPXRDR
2	The Data When Printed Will Resemble An OS/VS ABEND Dump.		DPPXRDR

Duplicate Data Set Support

One of the capabilities of the Special Real Time Operating System is to have a higher degree of direct access data reliability by duplicating some of the data sets. This functional area, called Duplicate Data Set (DDS), is a SYSGENed option and allows a limited degree of online data set switching.

During initialization, the frequently used DDS load modules are loaded into core and their addresses are saved within the DDS control table header.

The DDS data areas are constructed at DDS initialization time also, and DDS chains are established. There are four basic DDS data areas: The DDS control header (DDSCTLHD), the DDS control area (DDSCTLA), the DDS input/output area (DDSIOA) with its related DDS DECBs, and the DDS extension task chain (DDSXTCBC).

There are six logical chains in the DDS system which, along with the data areas, are graphically demonstrated in Figure 2-75.

The Systems Communication Vector Table (SCVT) has two pointers in it pertinent to DDS. The first pointer (SCVTDDSE at displacement 00) is to the DDSCTLHD, which is followed serially by all of the DDSCTLAs. Each DDSCTLA will point to a DDSIOA if a DDSDCB is currently DDS opened against this DDS.

The second pointer in the SCVT (SCVTDDSX at displacement 204_{10}) is to the first DDSXTCBC. Each DDSXTCBC will point to the next one, the last one pointing to zero. Examining the DDSXTCBCs will tell which tasks are using DDS in which ways.

The DDS routines can be divided into four major areas: control routines, subroutines, operator commands, and internal subroutine.

The control routines are responsible for initializing DDS, cleaning up open Data Control Blocks (DCBs) etc. at task termination, and ensuring that the proper data sets are in service, out-of-service, following a failover/restart.

The DDS subroutine provides the user interface with DDS during realtime execution while the DDS operator commands provide the operator communications and control over the DDS data sets (e.g., CREATE a back up, etc.).

The DDS internal subroutines provide the subservices required by each of the other three areas . They function as subroutines and may be called by one or more of the other routines in a number of different logical combinations.

COMPOSITE PICTORIAL DESCRIPTION OF ALL DDS CHAINS

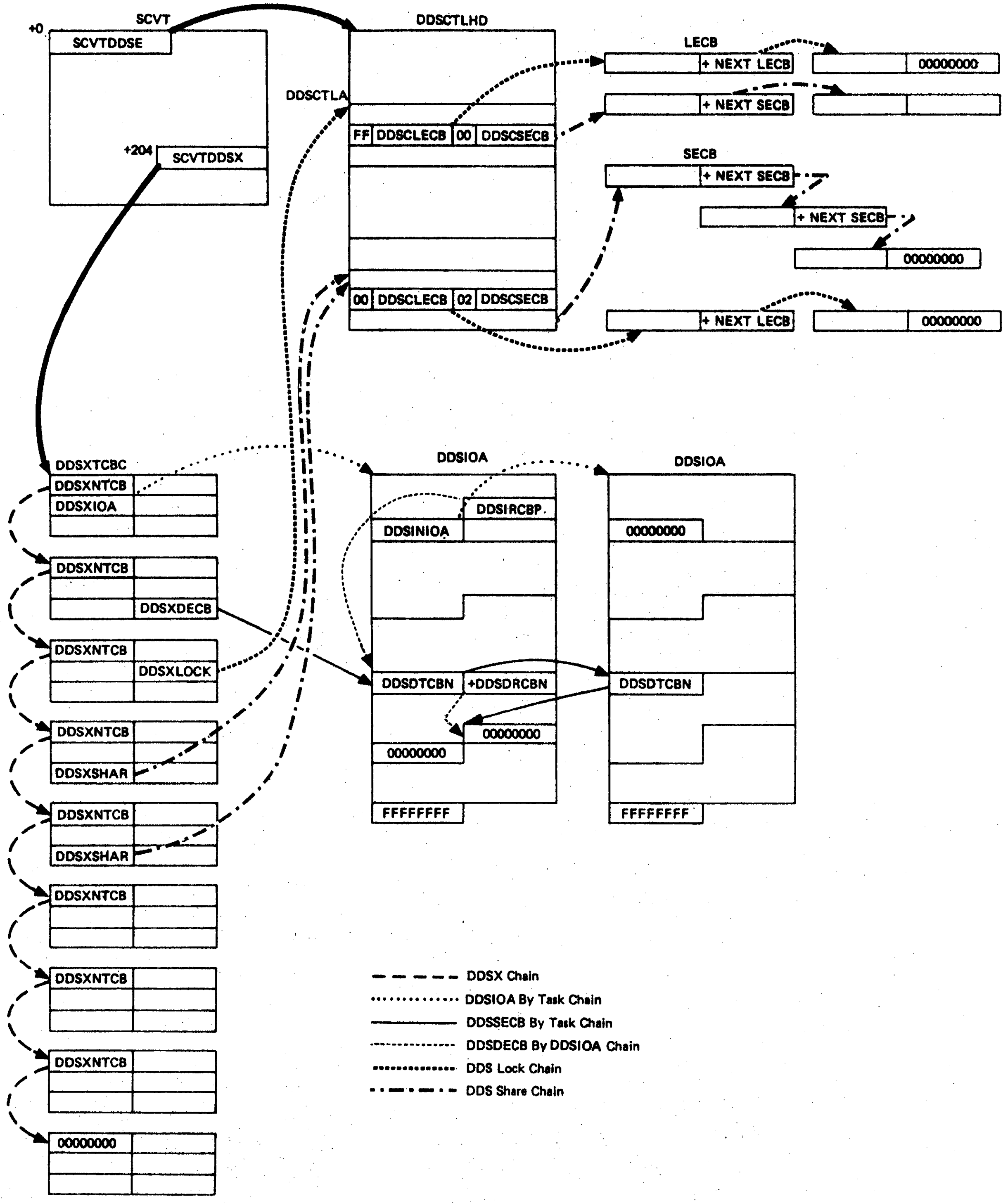
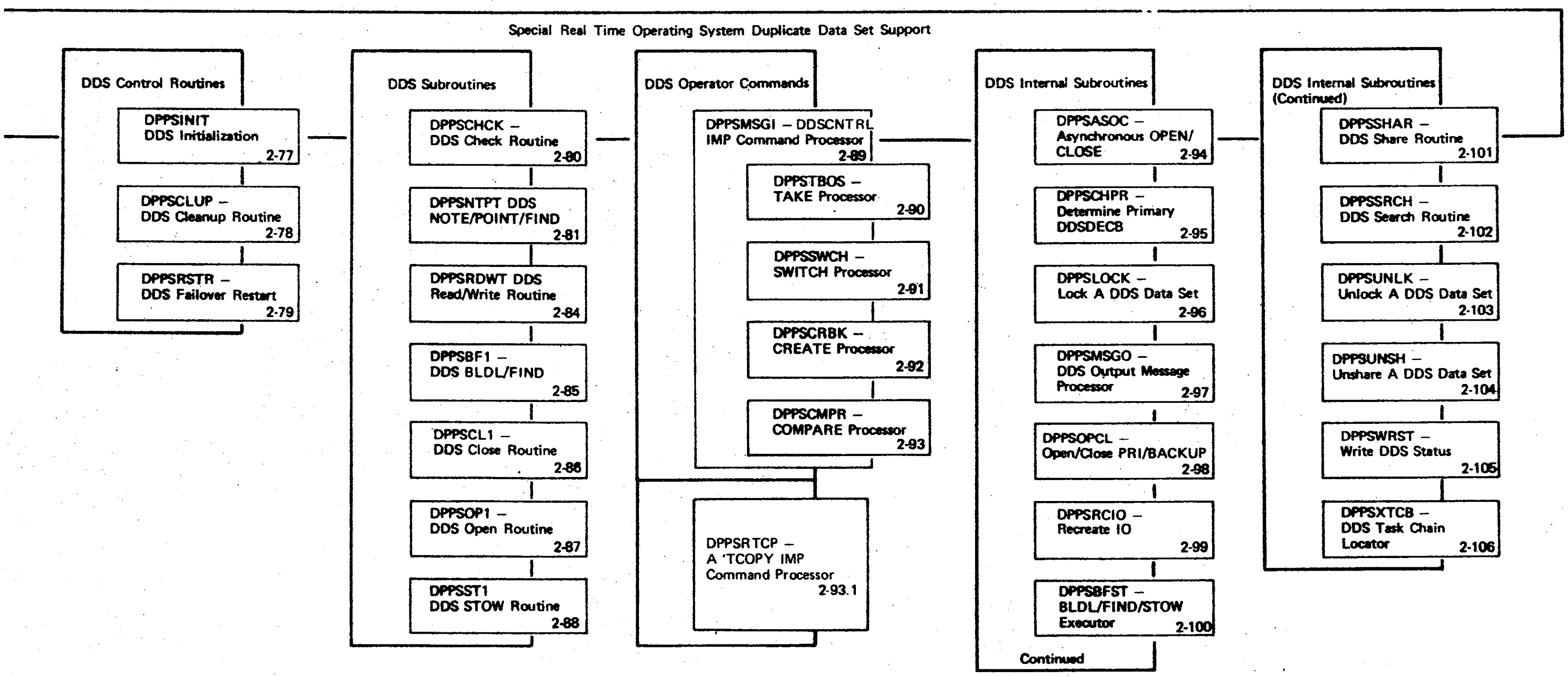


Figure 2-75 (1 Of 2) - Composite Pictorial Description Of All DDS Chains

	Descriptions
DDSX CHAIN	The SCVTDDSX Word Points To The First DDSXTCBC, Which Points To The Next DDSXTCBC, Etc. Each DDSXTCBC Consists Of Pointers Pertaining To A Different Task Using DDS.
DDSIOA By TASK CHAIN	All Of The DDSIOA's Opened By A Task Are Chained Together Via The DDSINIOA Word In The DDSIOA. The First DDSIOA Is Pointed To By The DDSXIOA Word In The DDSXTCBC For This Task.
DDSDECB By TASK CHAIN	Each DDSDECB Reserved By A Task Is Chained Through The DDSDTCBN Word Of The DDSDECB. The First Such DDSDECB Is Pointed To By The DDSXDECB Word Of The DDSXTCBC For That Task.
DDSDECB By DDSIOA CHAIN	Each Reserved DDSDECB For A DDSIOA Is Chained By The DDSDRCBN Word Of The DDSDECB. The First Reserved DDSDECB Is Pointed To By The DDSIRCBP Word Of The DDSIOA.
DDS LOCK CHAIN	Each Task Waiting To Lock A DDS Is Chained Together Via Its LECB. The First LECB Is Pointed To By The DDSCLECB Word Of The DDSCTLA. If A Task Has A DDA Locked, Its DDSXLOCK Word Of Its DDSXTCBC Will Point To The DDSCTLA.
DDS SHARE CHAIN	Each Task Waiting-To-Share A DDS Is Chained Off That DDS Via An SECB. The First SECB Is Pointed To By The DDSCSECB Word Of The DDSCTLA. If A Task Is Currently Sharing A DDS, Its DDSXSHAR Word Of Its DDSXTCBC Points To The DDSCTLA.

Figure 2-75, 2 Of 2

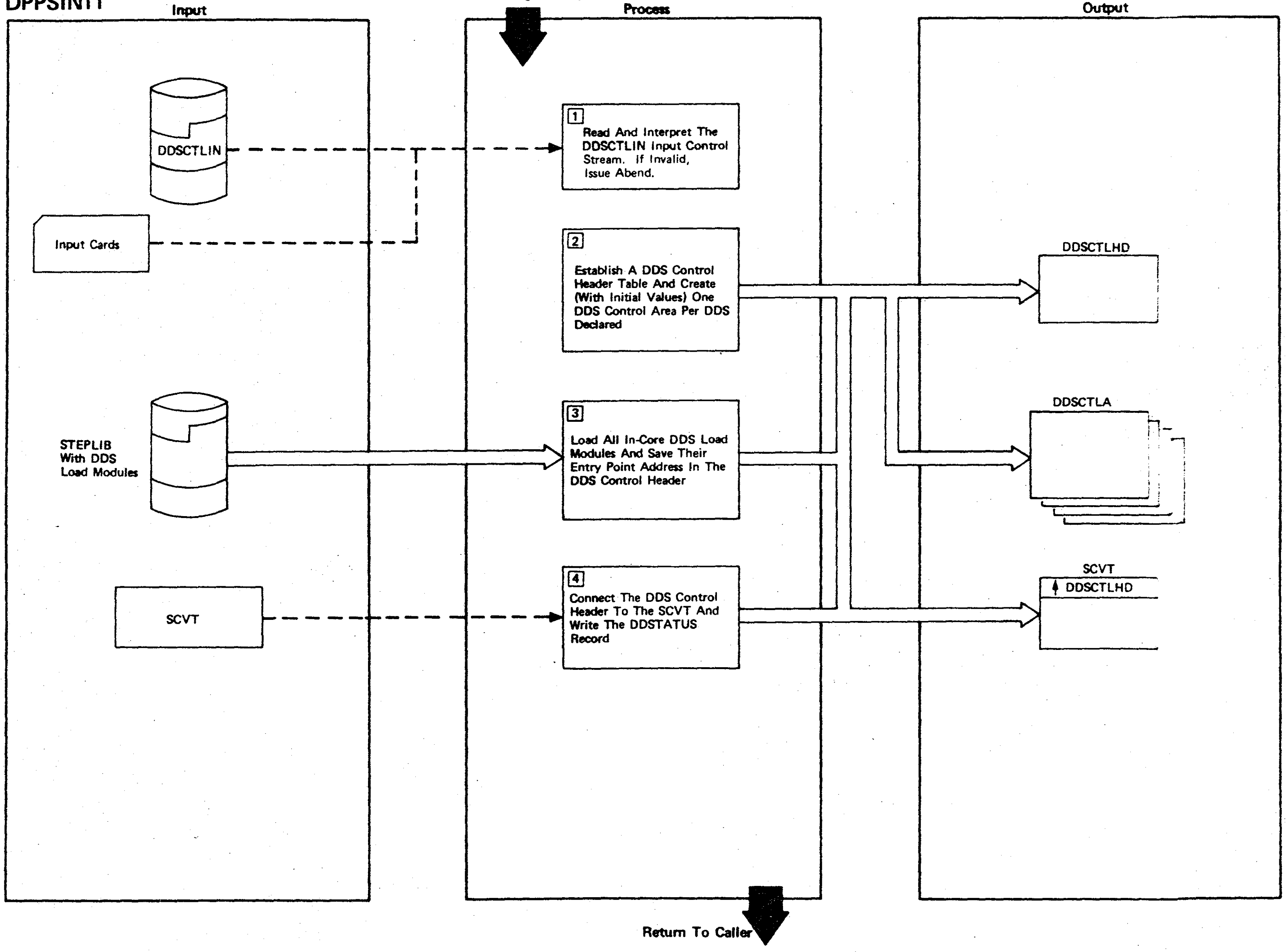

Special Real Time Operating System Duplicate Data Set Support
DDS Control Routines
DPPSINIT DDS Initialization 2-77
DPPSCLUP – DDS Cleanup Routine 2-78
DPPSRSTR – DDS Failover Restart 2-79
DDS Subroutines
DPPSCHCK – DDS Check Routine 2-80
DPPSNTPT DDS NOTE/POINT/FIND 2-81
DPPSRDWT DDS Read/Write Routine 2-84
DPPSBF1 – DDS BLDL/FIND 2-85
DPPSCL1 – DDS Close Routine 2-86
DPPSOP1 – DDS Open Routine 2-87
DPPSST1 DDS STOW Routine 2-88
DDS Operator Commands
DPPSMSGI – DDSCNTRL IMP Command Processor 2-89
DPPSTBOS – TAKE Processor 2-90
DPPSSWCH – SWITCH Processor 2-91
DPPSCRBK – CREATE Processor 2-92
DPPSCMPR – COMPARE Processor 2-93
DPPSRTCP – A 'TCOPY IMP Command Processor 2-93.1
DDS Internal Subroutines
DPPSASOC – Asynchronous OPEN/ CLOSE 2-94
DPPSCHPR – Determine Primary DDSDECB 2-95
DPPSLOCK – Lock A DDS Data Set 2-96
DPPSMSGO – DDS Output Message Processor 2-97
DPPSOPCL – Open/Close PRI/BACKUP 2-98
DPPSRCIO – Recreate IO 2-99
DPPSBFST – BLDL/FIND/STOW Executor 2-100
Continued
DDS Internal Subroutines (Continued)
DPPSSHAR – DDS Share Routine 2-101
DPPSSRCH – DDS Search Routine 2-102
DPPSUNLK – Unlock A DDS Data Set 2-103
DPPSUNSH – Unshare A DDS Data Set 2-104
DPPSWRST – Write DDS Status 2-105
DPPSXTCB – DDS Task Chain Locator 2-106

LICENSED MATERIAL — PROPERTY OF IBM

Figure 2-76 Special Real Time Operating System Duplicate Data Set Support Overview

DPPSINIT

From DPPINIT
(Figure 2-9, 9 Of 12)



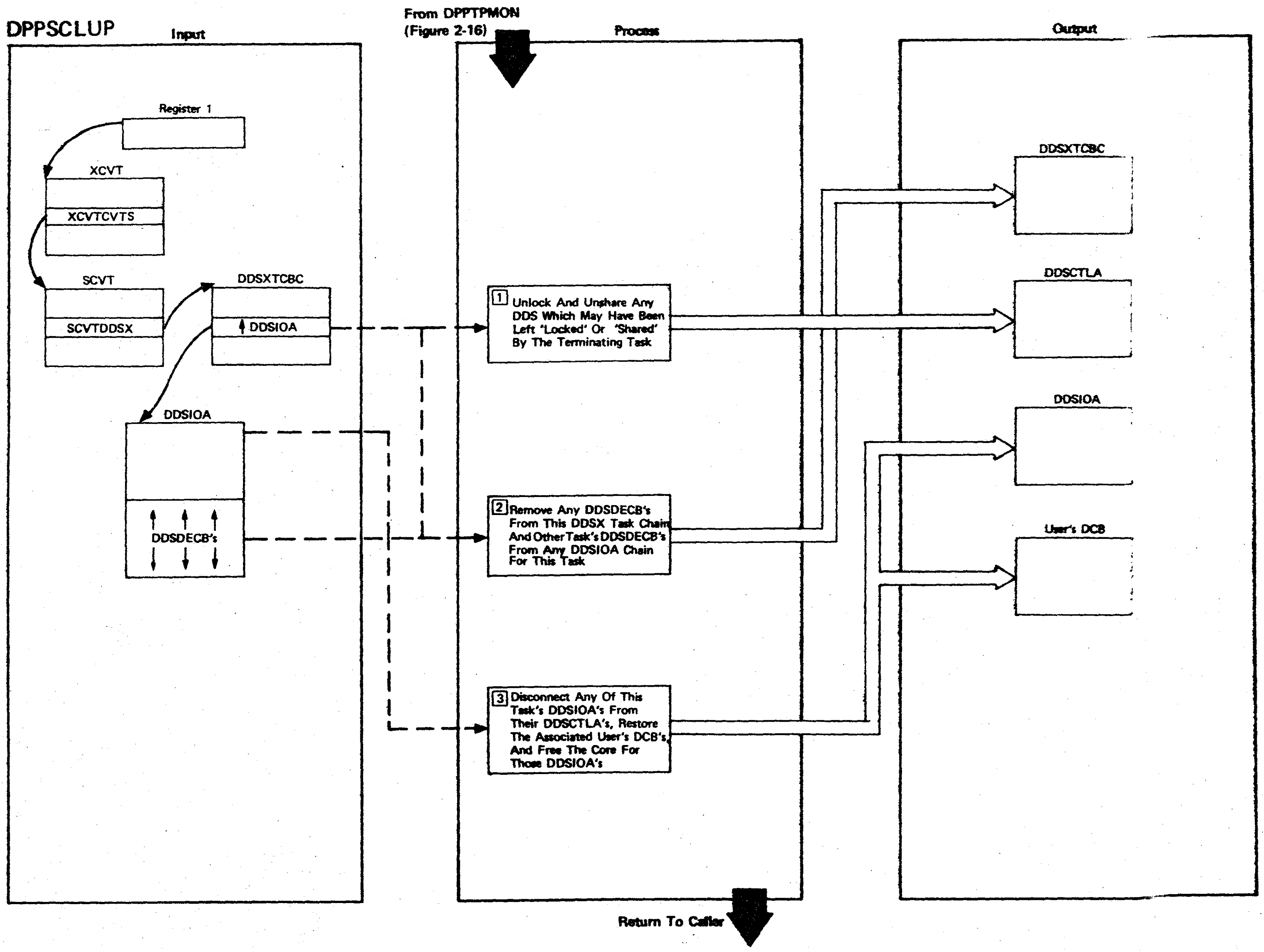
1. Read And Interpret The DDSCTLIN Input Control Stream. If Invalid, Issue Abend.
2. Establish A DDS Control Header Table And Create (With Initial Values) One DDS Control Area Per DDS Declared
3. Load All In-Core DDS Load Modules And Save Their Entry Point Address In The DDS Control Header
4. Connect The DDS Control Header To The SCVT And Write The DDSTATUS Record

Return To Caller

Figure 2-77 (1 Of 2) - DDS Initialization

Figure 2-77 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Normally, each card represents a DDS declaration (DDSNAMES card). If the first card is REFRESH or READONLY, read the DDSTATUS data set to determine the DDS declarations. If errors occur in conjunction with the DDS input control stream, ABEND with a decimal code of 80.	USER 80	DPPSINIT
2	The DDS control header (DDSCTLHD) and DDS control areas (DDSCTLA) will will be GETMAINed from subpool 0. A Special Real Time Operating System define lock will be needed for each DDSCTLA.		DPPSINIT
3	Except for the pseudo-SVC routines (DDSOPEN, DDSCLOSE, DDSFIND/BLDL, DDSSTOW), all other DDS load modules will be loaded and their entry points saved at predetermined slots in the DDSCTLHD.		DPPSINIT
4	Using the CHAIN function, the DDSCTLHD shall connect to the SCVT. This provides a pathway to the DDSCTLHD from the job step TCB through the TCBX, XCVT, SCVT. The DDSTATUS data set will keep the DDS declarations up to date. A message will be output if the DDSTATUS cannot be updated.	DPP881I	



1 Unlock And Unshare Any DDS Which May Have Been Left 'Locked' Or 'Shared' By The Terminating Task

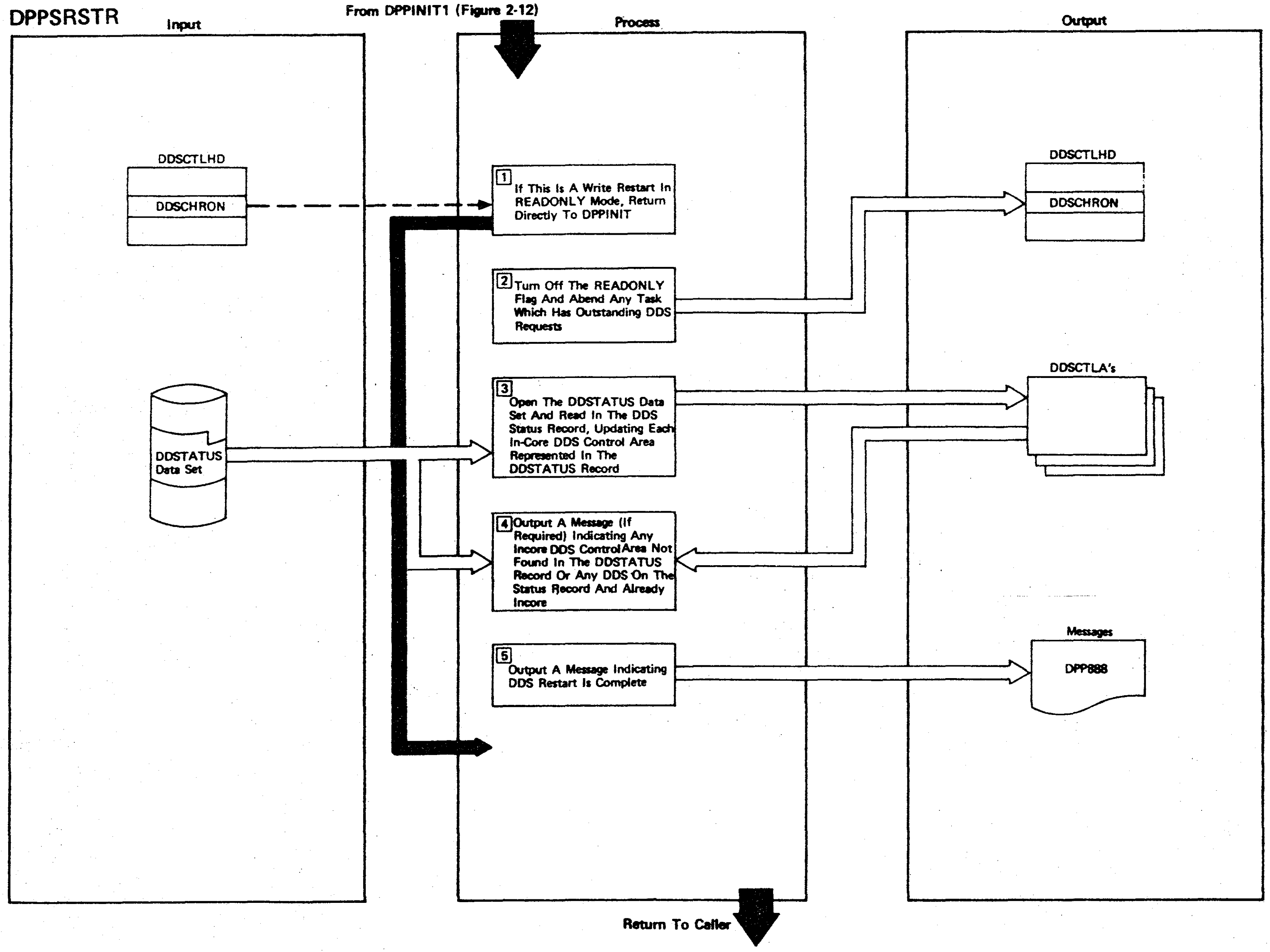
2 Remove Any DDSDECB's From This DDSX Task Chain And Other Task's DDSDECB's From Any DDSIOA Chain For This Task

3 Disconnect Any Of This Task's DDSIOA's From Their DDSCTLA's, Restore The Associated User's DCB's, And Free The Core For Those DDSIOA's

Figure 2-78 (1 Of 2) - DDS Task-End Cleanup

Figure 2-78 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Those DDSs which were left shared or locked, if any, will be in the DDSX task chain for this task, which can be located from the pointer to the DDSXTCBC within the SCVT.		DPPSCLUP
2	Since each reserved (unchecked) DDSDECB is logically part of two chains, the task chain and the DDSIOA chain, each must be de-chained from both chains if there are any for this task, or for any DDSIOA for this task.		DPPSCLUP
3	Any DDS opened DDSDCB will still be pending if they were not disclosed; therefore, these user DCBs must be restored to their preopen status. The OS/VS1 system will close the DCBs within the DDSIOA, which will also be freed.		DPPSCLUP



1 If This Is A Write Restart In READONLY Mode, Return Directly To DPPINIT

2 Turn Off The READONLY Flag And Abend Any Task Which Has Outstanding DDS Requests

3 Open The DDSTATUS Data Set And Read In The DDS Status Record, Updating Each In-Core DDS Control Area Represented In The DDSTATUS Record

4 Output A Message (If Required) Indicating Any Incore DDS Control Area Not Found In The DDSTATUS Record Or Any DDS On The Status Record And Already Incore

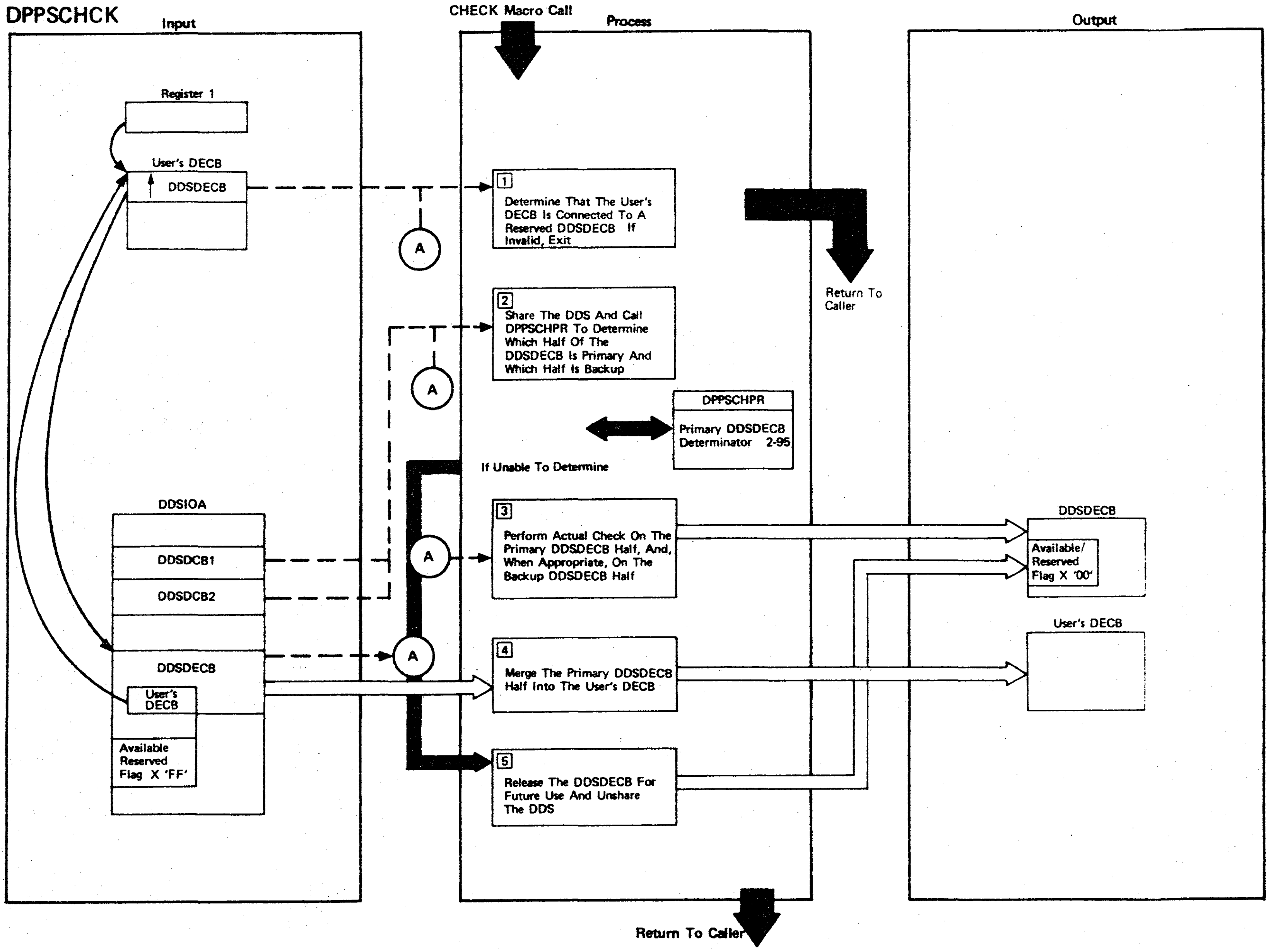
5 Output A Message Indicating DDS Restart Is Complete

Figure 2-79 (1 Of 2) - DDS Failover/Restart

Figure 2-79 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Failover/restart processing is not required at write restart data set time in read only mode.		DPPSRSTR
2	Those tasks which have outstanding DDS requested, as noted in the DDSX **task** chains, will ABEND with code decimal 81.	USER 81	DPPSRSTR
3	If the DDSTATUS data set cannot be opened or if the DDS status record cannot be read, output the appropriate messages.	DPP890I DPP891I DPP892I DPP882I	DPPSRSTR
4	Messages indicating that the DDS status data set has too many or not enough declarations alert the operator to these conditions.	DPP886I DPP887I	DPPSRSTR
5	The 'DDS restart is complete' message indicates that the DDS **aspect** of Failure/Restart (updating in-core DDS control tables) is completed.	DPP888I	DPPSRSTR

LICENSED MATERIAL — PROPERTY OF IBM



1 Determine That The User's DECB Is Connected To A Reserved DDSDECB If Invalid, Exit

2 Share The DDS And Call DPPSCHPR To Determine Which Half Of The DDSDECB Is Primary And Which Half Is Backup

3 Perform Actual Check On The Primary DDSDECB Half, And, When Appropriate, On The Backup DDSDECB Half

4 Merge The Primary DDSDECB Half Into The User's DECB

5 Release The DDSDECB For Future Use And Unshare The DDS

Figure 2-80 (1 Of 2) - DDS CHECK Module

Figure 2-80 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	This determination is made by comparing the user's DECB pointer in the DDSDECB with the address of the user's DECB. The availability/reserved flag (DDSDAVAL) within the DDSDECB should be X'FF' to indicate a READ/WRITE operation has been executed. If this condition is not satisfied, the user's SYNAD is taken, and control is returned to the user following his CHECK macro (bypassing steps 2-5).		DPPSCHCK
2	The module DPPSCHPR will return the address of the primary half of DDSDECB and the backup half. If the DDSDECB is not properly organized and such a determination cannot be made, the user's SYNAD will be taken and control will proceed directly to step 5.		DPPSCHCK
3	After initially checking on the primary half DDSDECB, the backup half is checked if both the backup is in-service and the DCB is opened for update or output.		DPPSCHCK
	If SYNAD occurs on the primary DDSDECB and indicates a hardware failure, switchover will occur automatically if the backup is in-service. If SYNAD occurs during the backup CHECK, the backup is taken out-of-service. If SYNAD occurs and there is no backup or no hardware failure, the user's SYNAD is taken.		
4	The user's DECB will contain the correct ECB code and pointers to the primary DDSDECB data (IOB, etc.). When running in update mode and the check is for a prior read operation, the backup IOB address is saved in the user's DECB in the low-order three bytes of his ECB.		DPPSCHCK
5	The availability flag is set to X'00' and the DDSDECB is taken off its DDSIOA reserved and TASK reserved chains.		DPPSCHCK

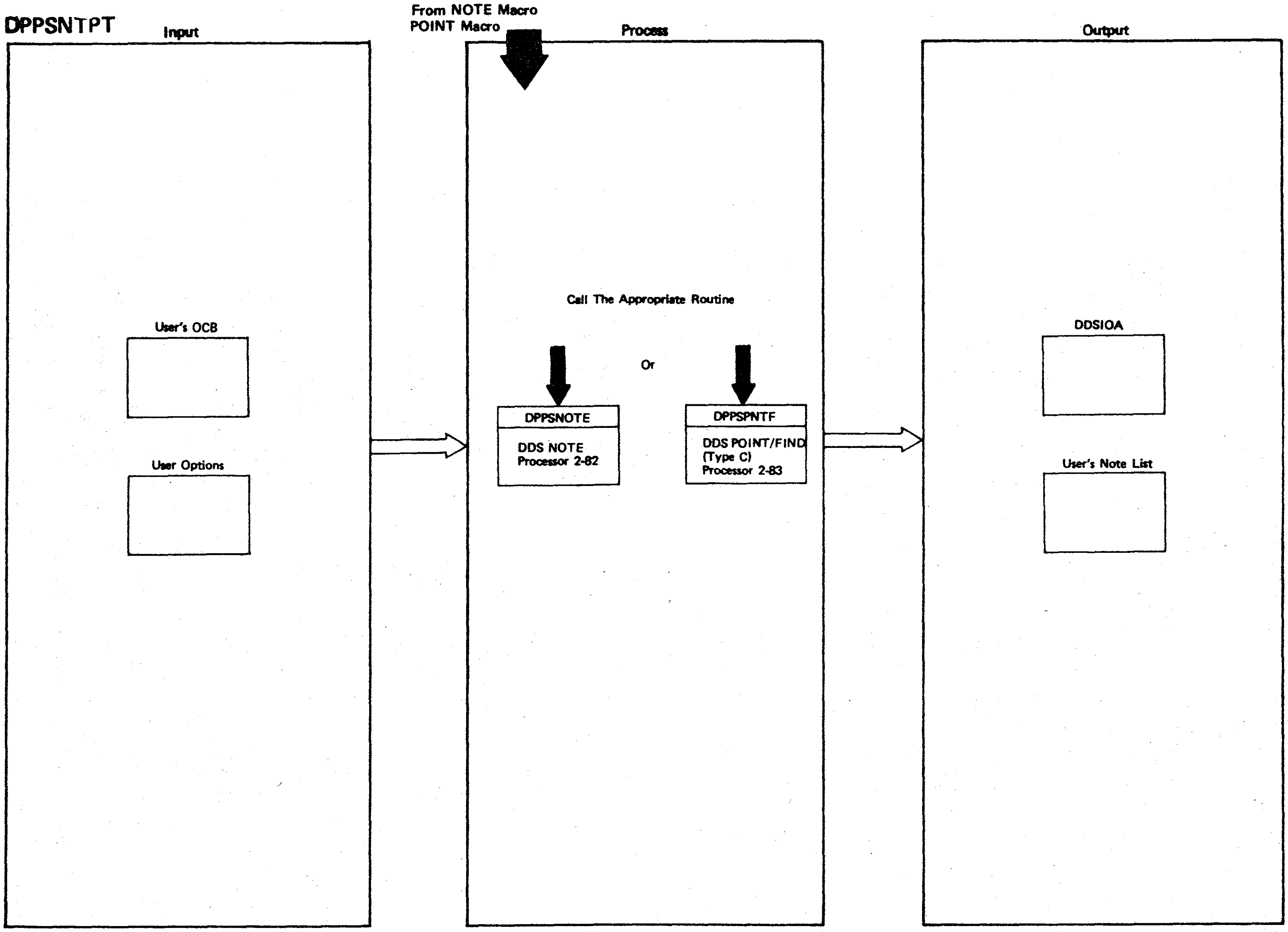


Figure 2-81 - DDS NOTE/POINT Sending Interface

Intentionally Blank

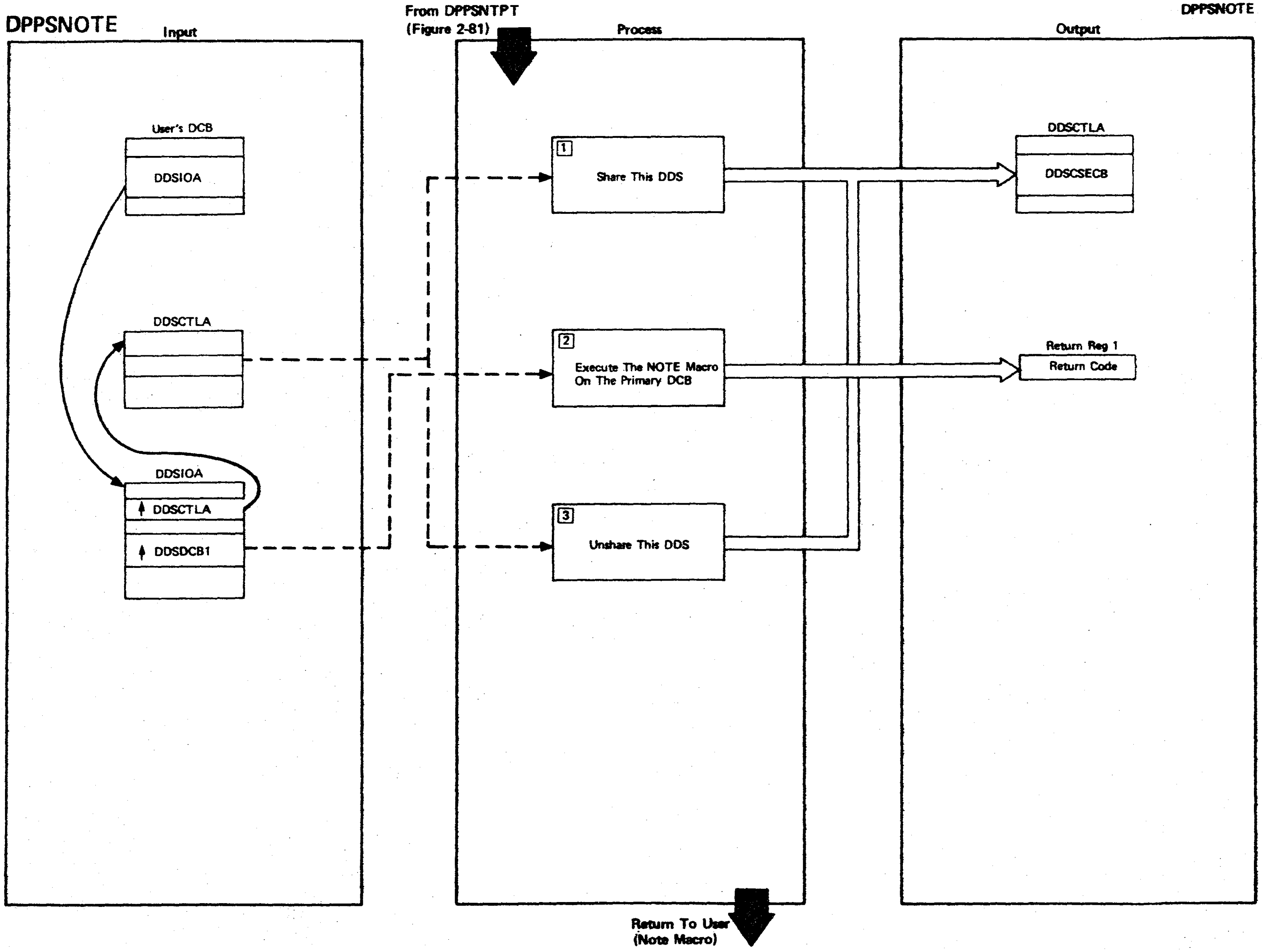


Figure 2-82 (1 Of 2) - DDS NOTE Internal

Figure 2-82 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The DDS share function is used to prevent a lockout of this DDS during the note operation.		DPPSNOTE
2	The primary DCB is used since there is no output, and the backup need not be inspected.		DPPSNOTE
3	This DDS is released for other tasks to lock.		DPPSNOTE

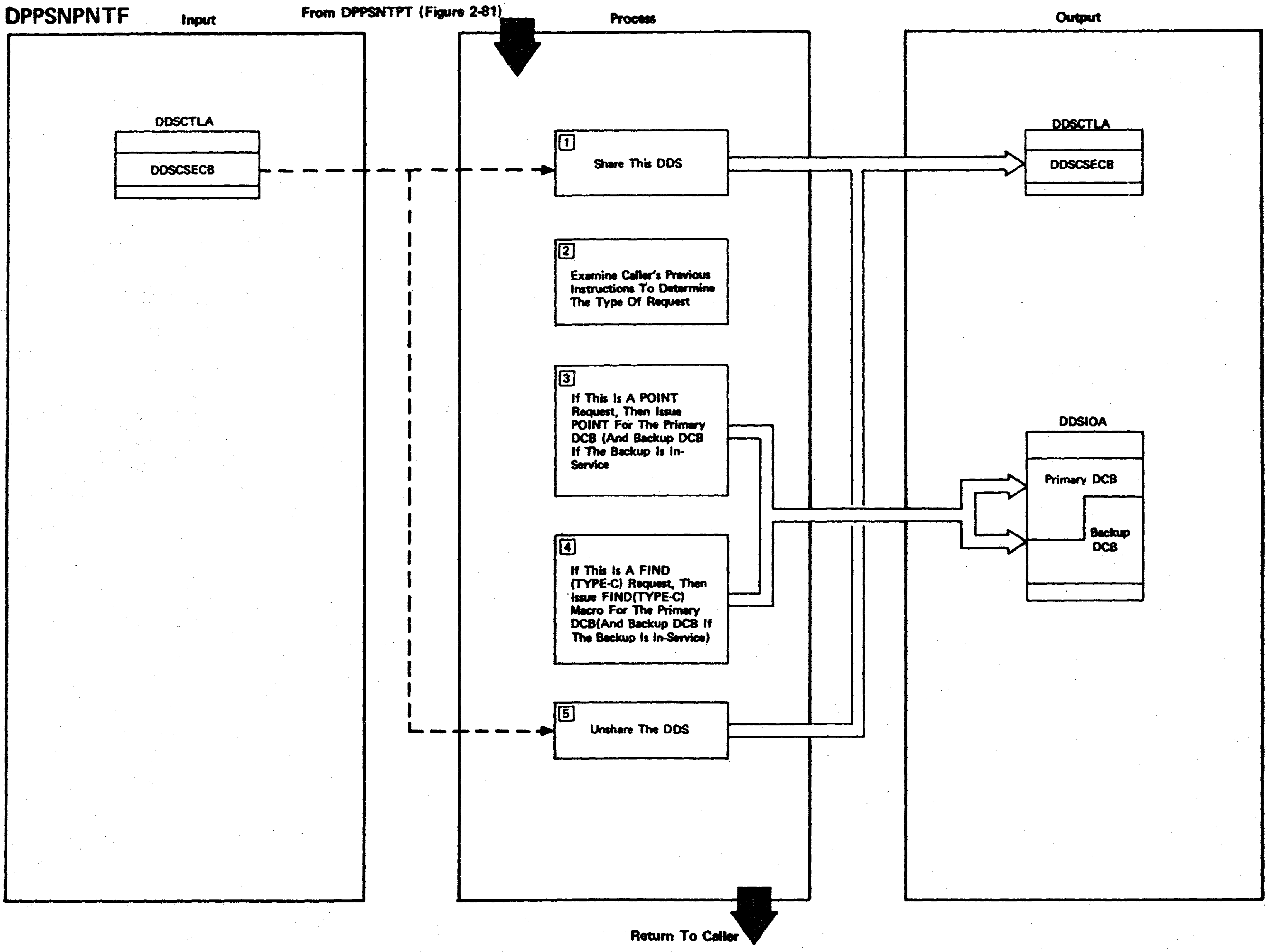

DPPSNPNTF
Input
From DPPSNTPT (Figure 2-81)
Process
Output
DDSCTLA
DDSCSECB
1 Share This DDS
2 Examine Caller's Previous Instructions To Determine The Type Of Request
3 If This Is A POINT Request, Then Issue POINT For The Primary DCB (And Backup DCB If The Backup Is In-Service
4 If This Is A FIND (TYPE-C) Request, Then Issue FIND(TYPE-C) Macro For The Primary DCB(And Backup DCB If The Backup Is In-Service)
5 Unshare The DDS
DDSCTLA
DDSCSECB
DDSIOA
Primary DCB
Backup DCB
Return To Caller

Figure 2-83 (1 Of 2) - DDS POINT Or FIND (Type-C)

Figure 2-83 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The share function will be used to prevent a lockout of this DDS until to POINT or FIND is completed.		DPPSPNTF
2	The indexing on the BALR instruction within the macro indicates the type of macro (no index = POINT, index = FIND (TYPE = C)).		DPPSPNTF
3	The POINT should be executed to the backup also if the backup is in-service.		DPPSPNTF
4	The FIND (TYPE = C) should be executed for the backup when the backup is in-service.		DPPSPNTF
5	The share function will release this DDS for locking.		DPPSPNTF

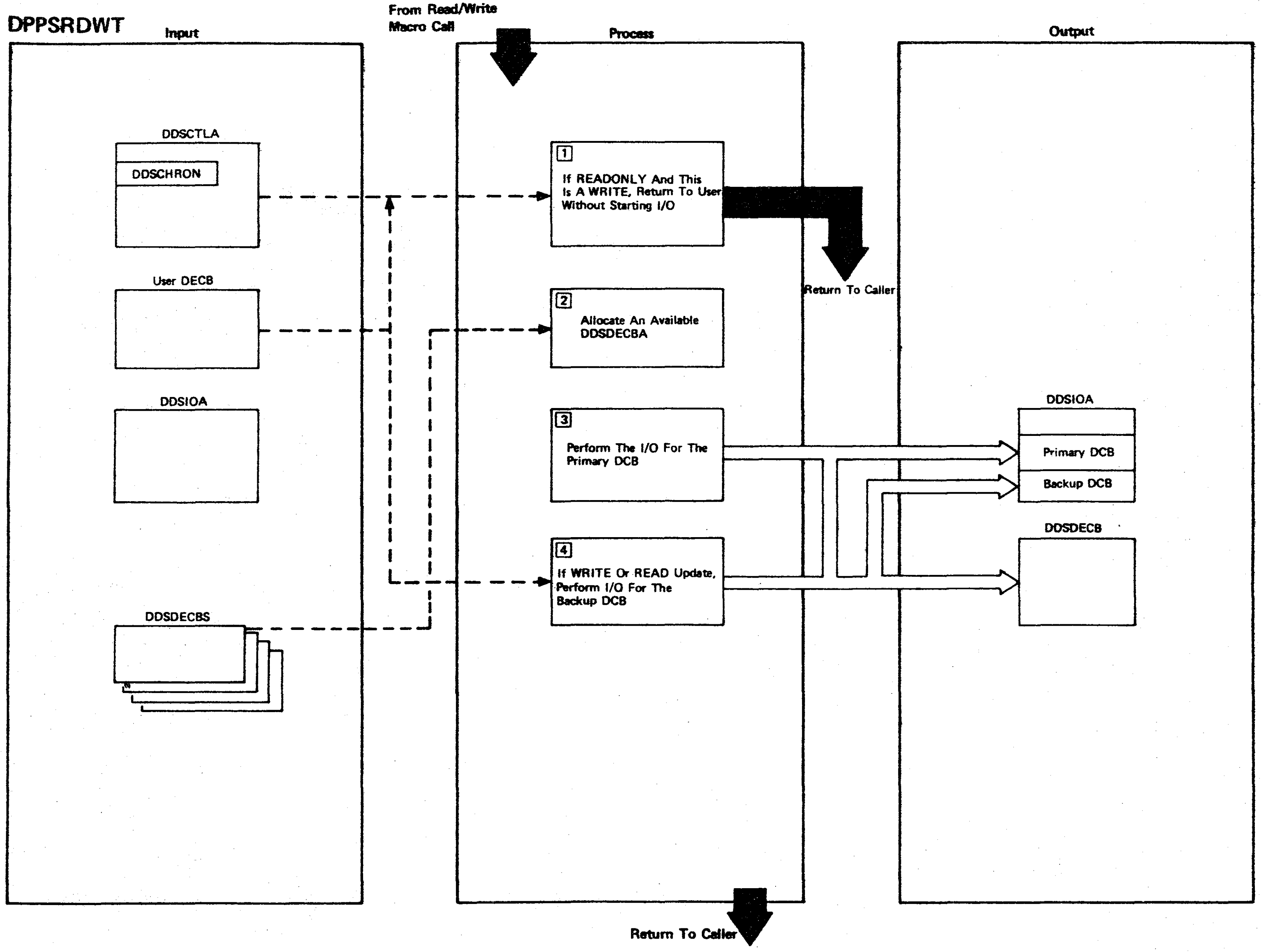

DPPSRDWT
Input
From Read/Write Macro Call
Process
Output
DDSCTLA
DDSCHRON
User DECB
DDSIOA
DDSDECBS
1
If READONLY And This Is A WRITE, Return To User Without Starting I/O
Return To Caller
2
Allocate An Available DDSDECBA
3
Perform The I/O For The Primary DCB
4
If WRITE Or READ Update, Perform I/O For The Backup DCB
DDSIOA
Primary DCB
Backup DCB
DDSDECB
Return To Caller

Figure 2-84 (1 Of 2) - DDS READ/WRITE Module

Figure 2-84 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Output operations are invalid when running in read only mode. Returning to user without starting I/O will cause a SYNAD if he should try the related check, so a meaningful return code is given.		DPPSRDWT
2	Should no DDSDECBA be available for I/O the user will be given a meaningful return code.		DPPSRDWT
	When an available DDSDECBA is found, it is marked reserved and connected to the user's DECB.		
3	The I/O is started against the primary DCB by branching to the appropriate OS/VS1 routine.		DPPSRDWT
4	The I/O needs to be started against the backup if in UPDATE mode so that the subsequent write update can be done for the backup also.		DPPSRDWT

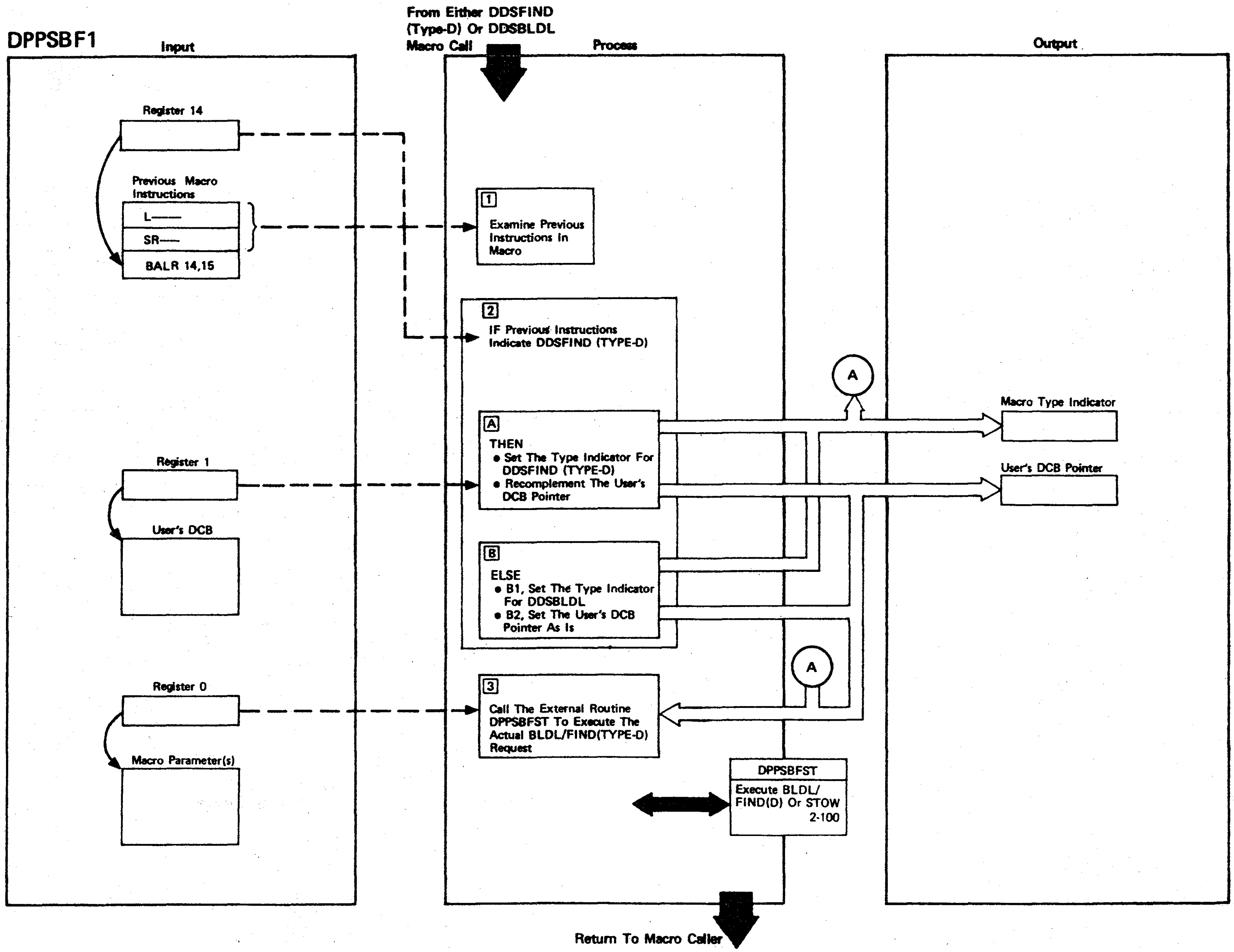

DPPSBF1
Input
From Either DDSFIND (Type-D) Or DDSBLDL Macro Call
Process
Output
Register 14
Previous Macro Instructions
L——
SR——
BALR 14,15
1
Examine Previous Instructions In Macro
2
IF Previous Instructions Indicate DDSFIND (TYPE-D)
A
A
THEN
• Set The Type Indicator For DDSFIND (TYPE-D)
• Recomplement The User's DCB Pointer
Macro Type Indicator
User's DCB Pointer
Register 1
User's DCB
B
ELSE
• B1, Set The Type Indicator For DDSBLDL
• B2, Set The User's DCB Pointer As Is
A
Register 0
3
Call The External Routine DPPSBFST To Execute The Actual BLDL/FIND(TYPE-D) Request
Macro Parameter(s)
DPPSBFST
Execute BLDL/ FIND(D) Or STOW
2-100
Return To Macro Caller

Figure 2-85 (1 Of 2) - BLDL/FIND (Type-D) Formatter

Figure 2-85 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The previous instructions generated by the DDSBLDL or DDSFIND (Type-D) macros indicate which type of macro was issued (an LCR 1, 1 instruction would indicate DDSFIND instead of DDSBLDL).		DPPSBF1
2	The type indicator (used as input to DPPSBFST) is 0 for BLDL and 4 for FIND (TYPE-D).		DPPSBF1
	The user's DCB pointer will have been complemented (negative value) by the macro instructions for DDS, FIND (TYPE-D).		
3	The external routine DPPSBFST executes the actual BLDL or FIND and expects as input the user's DCB, the macro parameter, and the type. The return code from DPPSBFST is not altered when returning to the user.		DPPSBF1

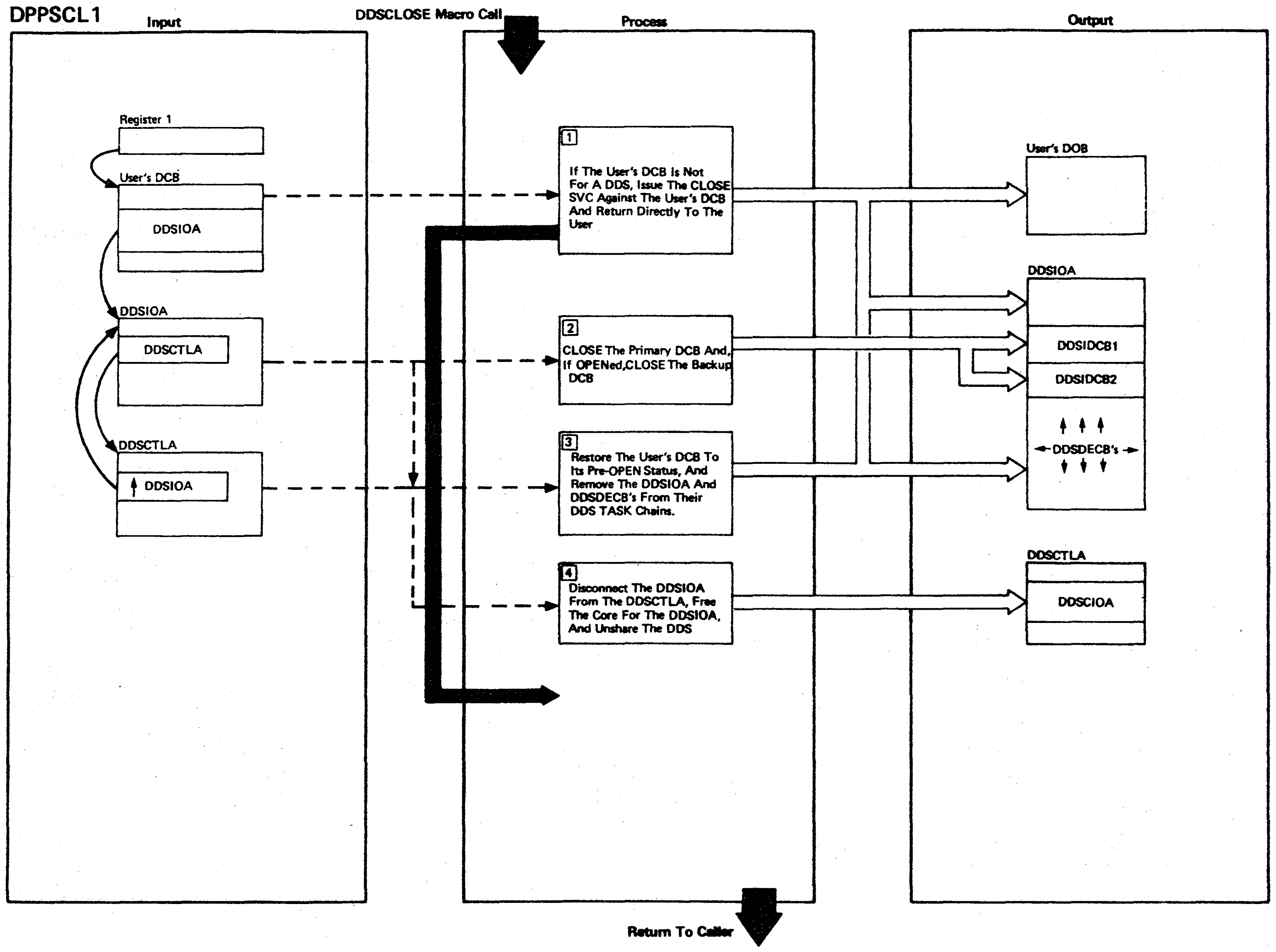

DPPSCL1
Input
DDSCLOSE Macro Call
Process
Output
Register 1
User's DCB
DDSIOA
DDSIOA
DDSCTLA
DDSCTLA
DDSIOA
1
If The User's DCB Is Not For A DDS, Issue The CLOSE SVC Against The User's DCB And Return Directly To The User
2
CLOSE The Primary DCB And, If OPENed,CLOSE The Backup DCB
3
Restore The User's DCB To Its Pre-OPEN Status, And Remove The DDSIOA And DDSDECB's From Their DDS TASK Chains.
4
Disconnect The DDSIOA From The DDSCTLA, Free The Core For The DDSIOA, And Unshare The DDS
User's DOB
DDSIOA
DDSIDCB1
DDSIDCB2
DDSDECB's
DDSCTLA
DDSCIOA
Return To Caller

Figure 2-86 (1 Of 2) - Close A DDSDCB

Figure 2-86 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the user did not declare this DDS during initialization, it will be running in single mode, and the user's DCB will be closed. Steps 2 through 4 should be omitted, and control returned directly to the user.		DPPSCLI
2	The DDS must be shared and both the primary DCB (within the DDSIOA) and the backup DCB should be closed, if the backup had been in-service at open time.		DPPSCLI
3	Those fields of the user's DCB (DDSIOA, DDS READ/WRITE, DDSCHECK, DDSNOTE/POINT, DCBOFLGS) which were altered at DDSOPEN time will be restored to their pre-open status. The DDSIOA and any unchecked DDSDECB for this DDSIOA will be removed from their DDSX task chain (DDSXTCBC) which is created the first time a task uses DDS.		DPPSCLI
4	The DDSIOA pointer in the DDSCTLA will be zeroed and the core for this DDSIOA will be FREEMAINed, the FREEMAIN parameters having been saved at DDSOPEN time. This DDS will be unshared.		DPPSCLI

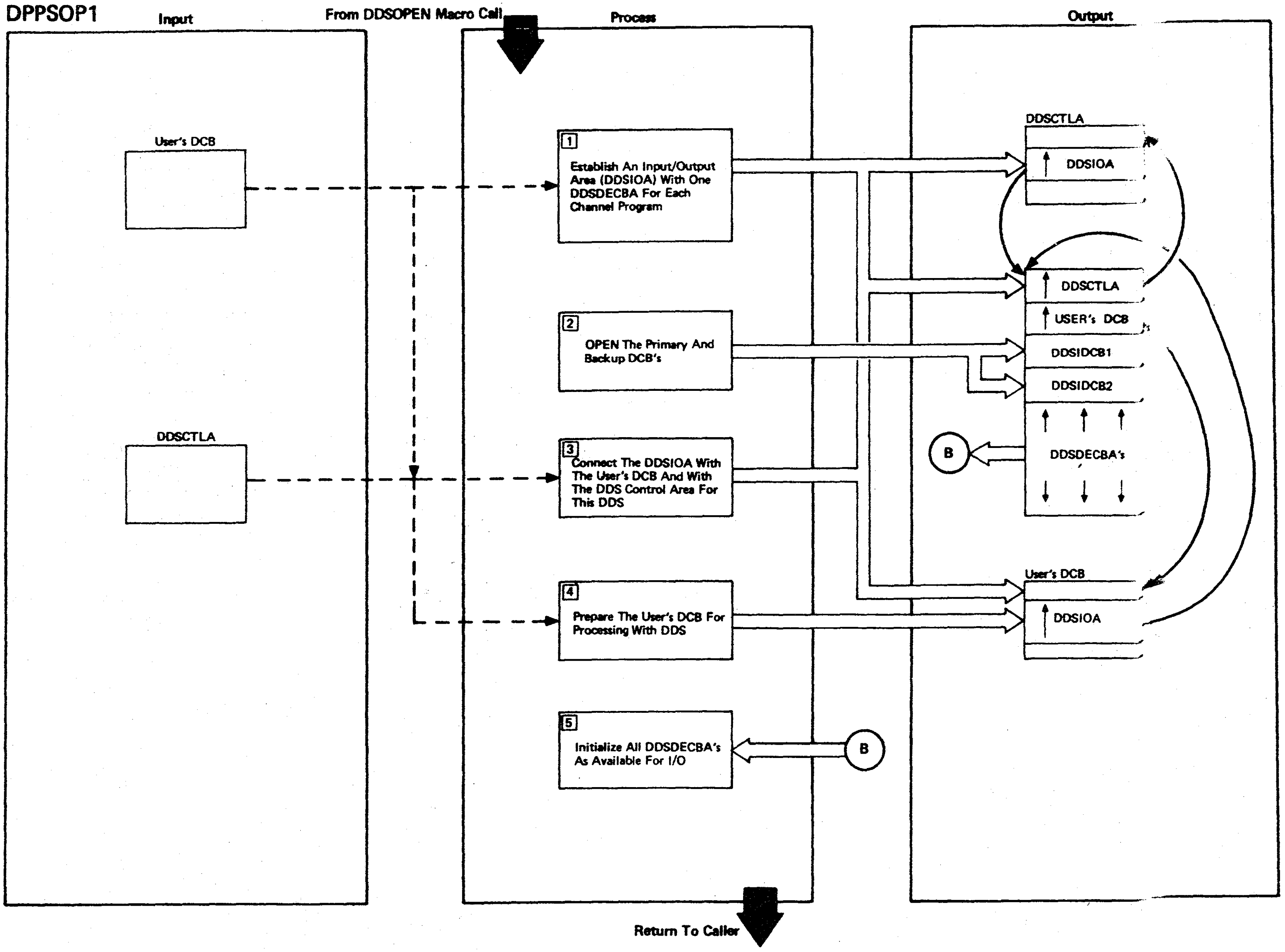

DPPSOP1
Input
From DDSOPEN Macro Call
Process
Output
User's DCB
DDSCTLA
1
Establish An Input/Output Area (DDSIOA) With One DDSDECBA For Each Channel Program
2
OPEN The Primary And Backup DCB's
3
Connect The DDSIOA With The User's DCB And With The DDS Control Area For This DDS
4
Prepare The User's DCB For Processing With DDS
5
Initialize All DDSDECBA's As Available For I/O
B
DDSCTLA
DDSIOA
DDSCTLA
USER's DCB
DDSIDCB1
DDSIDCB2
DDSDECBA's
B
User's DCB
DDSIOA
Return To Caller

Figure 2-87 (1 Of 2) - DDS OPEN Routine

Figure 2-87 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The DDSIOA will be GETMAINED from subpool 0 and the FREEMAIN parameters saved. The NCP operand (default = 1) of the user's DCB will determine how many DDSDECB pairs are generated.		DPPSOP1
2	If the primary OPEN fails, automatic switching will be performed if the backup is in-service. If the OPEN on the backup fails, take backup out-of-service will become automatic.		DPPSOP1
3	The DDSCTLA should point to the DDSIOA and vice versa. The user's DCB should point to the DDSIOA and vice versa.		DPPSOP1
4	The DCBOFLGS must be set and the addresses of DDSREAD/WRITE, DDSNOTE/POINT, and DDSCHECK must be set in the user's DCB.		DPPSOP1
5	Each DDSDECBA will be initialized and available for use in I/O requests. There will be a serial chain of available DDSDECBs minus the reserved DDSDECBs.		DPPSOP1

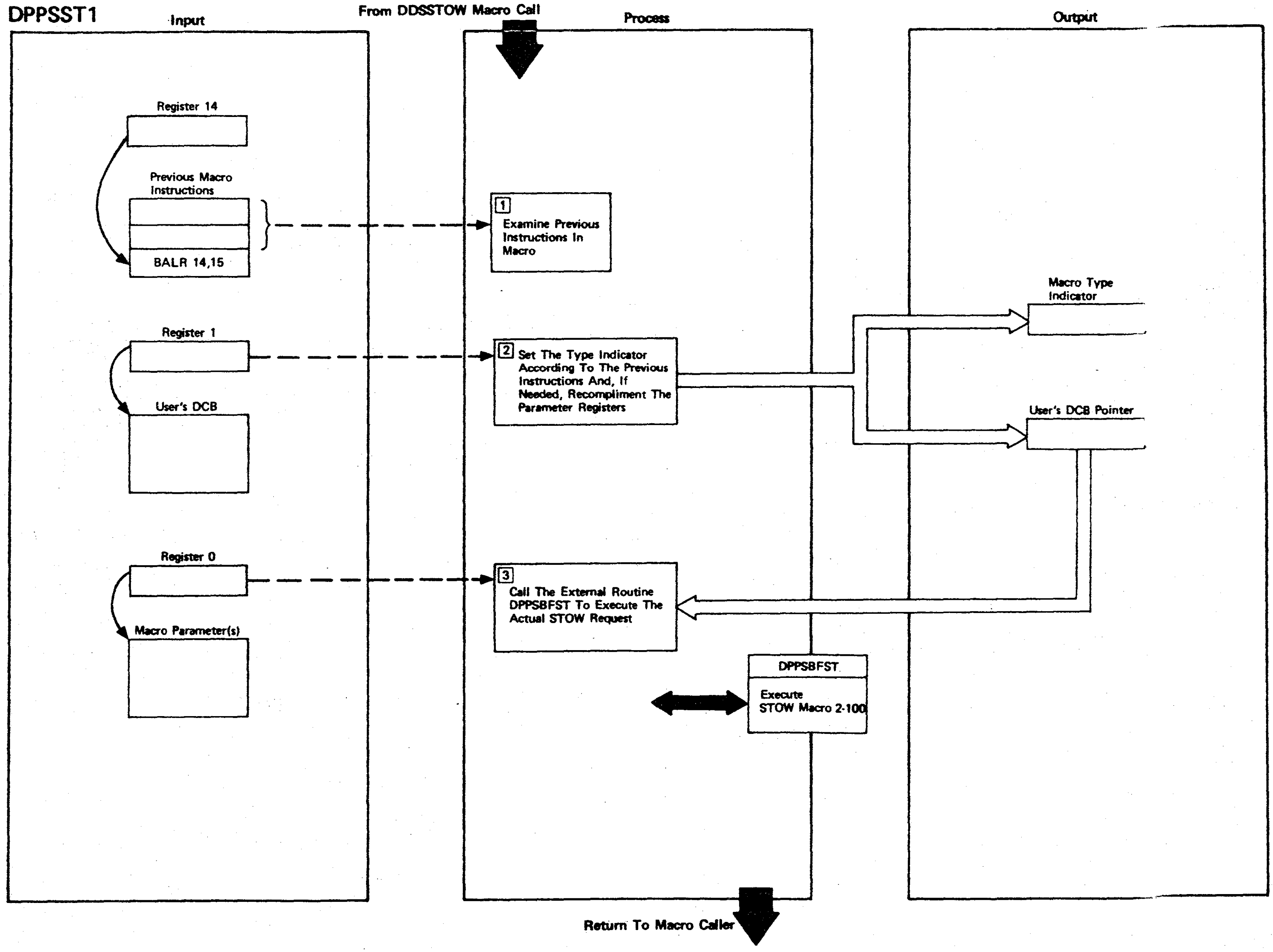

DPPSST1
Input
From DDSSTOW Macro Call
Process
Output
Register 14
Previous Macro Instructions
BALR 14,15
1 Examine Previous Instructions In Macro
Register 1
User's DCB
2 Set The Type Indicator According To The Previous Instructions And, If Needed, Recompliment The Parameter Registers
Macro Type Indicator
User's DCB Pointer
Register 0
Macro Parameter(s)
3 Call The External Routine DPPSBFST To Execute The Actual STOW Request
DPPSBFST
Execute STOW Macro 2-100
Return To Macro Caller

Figure 2-88 (1 Of 2) - DDS STOW Formatter

Figure 2-88 (2 of 2).

<table>
<tr><th>Step</th><th>Extended Description</th><th>Messages and ABEND Codes</th><th>PDL Segment</th></tr>
<tr><td>1</td><td>The previous instructions of the STOW macro indicate which types of STOW macro is being issued.</td><td></td><td>DPPSST1</td></tr>
<tr><td>2</td><td>Set up type indicators by complementing registers as follows:
<table>
<tr><th>Macro Type</th><th>Register to Complement</th></tr>
<tr><td>STOW (R)</td><td>1</td></tr>
<tr><td>STOW (D)</td><td>0</td></tr>
<tr><td>STOW (C)</td><td>0 and 1</td></tr>
<tr><td>STOW (A)</td><td>none</td></tr>
</table></td><td></td><td>DPPSST1</td></tr>
<tr><td>3</td><td>The external routine DPPSBFST will execute the actual STOW macro to both primary and backup (if in service) and returns the proper return code for the macro caller.</td><td></td><td>DPPSST1</td></tr>
</table>

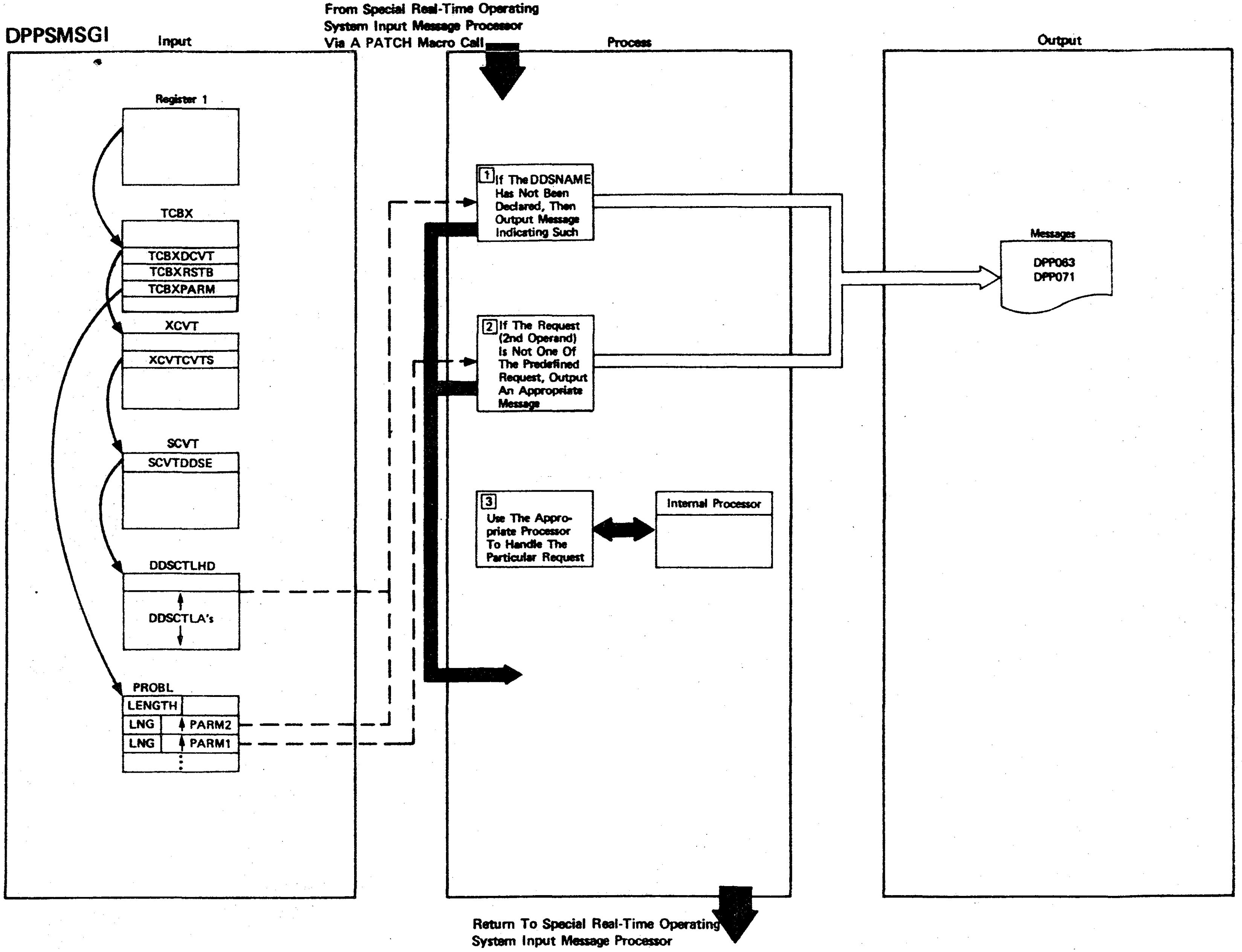

DPPSMSGI
Input
From Special Real-Time Operating System Input Message Processor Via A PATCH Macro Call
Process
Output
Register 1
TCBX
TCBXDCVT
TCBXRSTB
TCBXPARM
XCVT
XCVTCVTS
SCVT
SCVTDDSE
DDSCTLHD
DDSCTLA's
PROBL
LENGTH
LNG
PARM2
LNG
PARM1
1 If The DDSNAME Has Not Been Declared, Then Output Message Indicating Such
2 If The Request (2nd Operand) Is Not One Of The Predefined Request, Output An Appropriate Message
3 Use The Appropriate Processor To Handle The Particular Request
Internal Processor
Messages
DPP063
DPP071
Return To Special Real-Time Operating System Input Message Processor

Figure 2-89 (1 Of 14) - DDS Input Message Processor Main Segment

Figure 2-89 (2 of 14).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The message DDS NOT DECLARED will be output when the DDSNAME is not declared in the DDSCTLAs.	DPP063I	DPPSMSGI
2	Valid requests are STATUS, TAKE, CREATE, COMPARE, REPLACE, and SWITCH. The default is STATUS, and if the code is none of these, output the message DDS REQUEST NOT UNDERSTOOD.	DPP071I	DPPSMSGI
3	There is a processor HIPO chart for each internal processor: TAKE (see Figure 2-89 (3 of 14)) STATUS (see Figure 2-89 (5 of 14)) SWITCH (see Figure 2-89 (7 of 14)) CREATE (see Figure 2-89 (9 of 14)) REPLACE (see Figure 2-89 (11 of 14)) COMPARE (see Figure 2-89 (13 of 14))		DPPSMSGI

DPPSMSGI (Take)

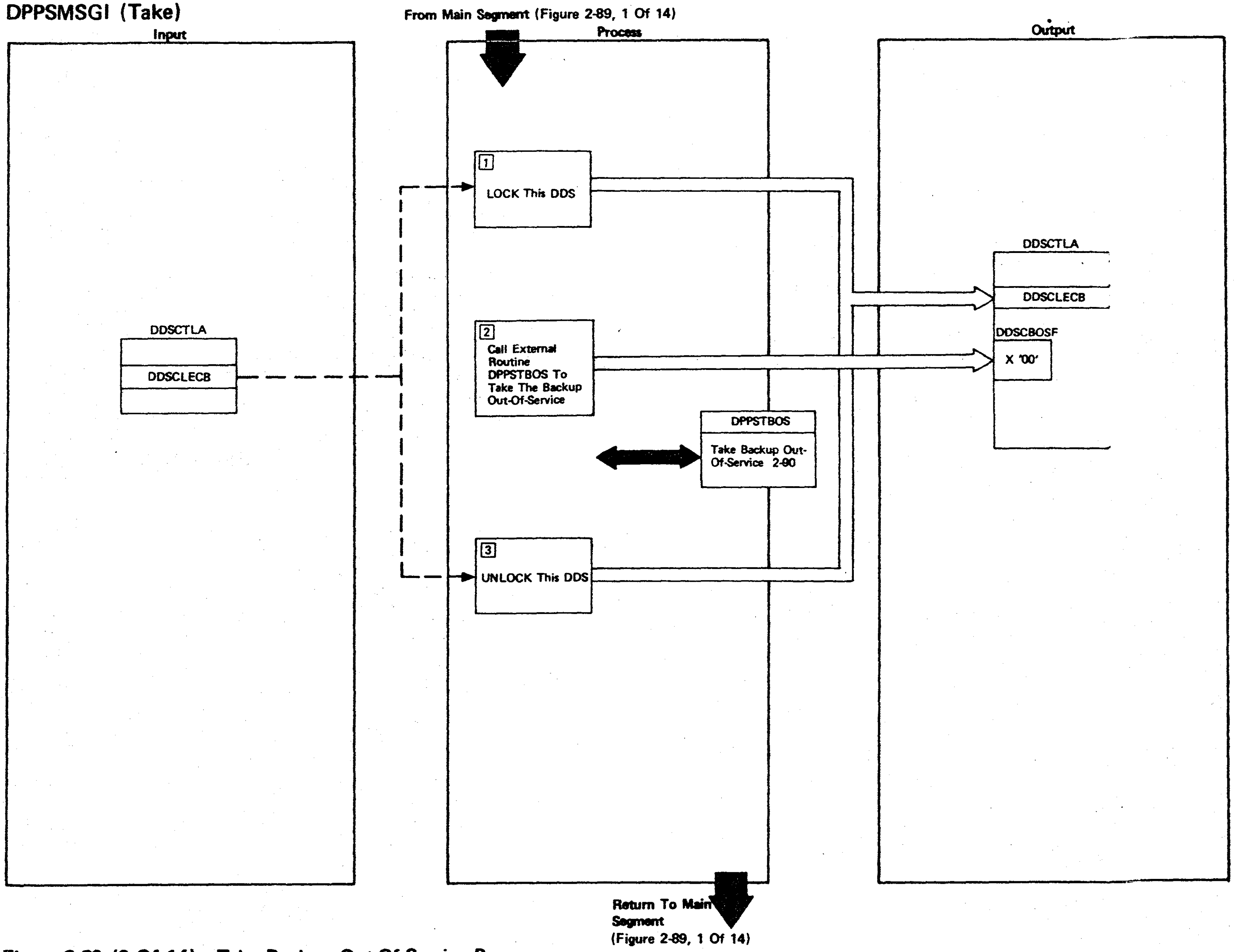

From Main Segment (Figure 2-89, 1 Of 14)
Input
Process
Output
DDSCTLA
DDSCLECB
1
LOCK This DDS
2
Call External Routine DPPSTBOS To Take The Backup Out-Of-Service
DPPSTBOS
Take Backup Out-Of-Service 2-90
3
UNLOCK This DDS
DDSCTLA
DDSCLECB
DDSCBOSF
X '00'
Return To Main Segment (Figure 2-89, 1 Of 14)

Figure 2-89 (3 Of 14) - Take Backup Out-Of-Service Processor

Figure 2-89 (4 of 14).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Use the internal DDS LOCK function to inhibit the use by other tasks of this DDS until the function is completed.		DPPSMSGI
2	The routine DPPSTBOS will cause the backup to be taken out-of-service (if not already) and will set the DDSCTLA as such.		DPPSMSGI
3	Use the internal DDS UNLOCK function to release the LOCK on this DDS.		DPPSMSGI

DPPSMSGI (Status)

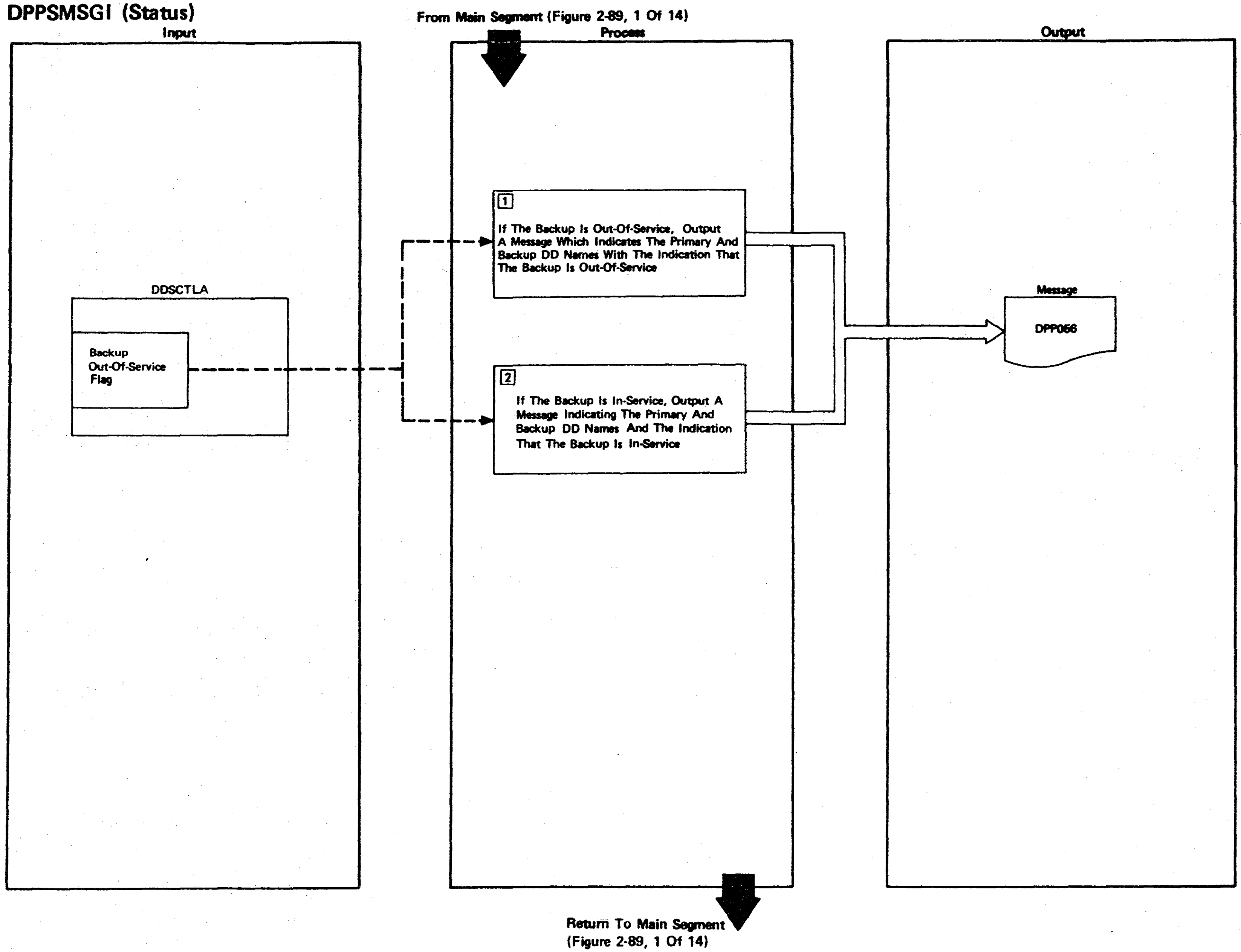

From Main Segment (Figure 2-89, 1 Of 14)
Input
Process
Output
DDSCTLA
Backup Out-Of-Service Flag
1
If The Backup Is Out-Of-Service, Output A Message Which Indicates The Primary And Backup DD Names With The Indication That The Backup Is Out-Of-Service
2
If The Backup Is In-Service, Output A Message Indicating The Primary And Backup DD Names And The Indication That The Backup Is In-Service
Message
DPP056
Return To Main Segment (Figure 2-89, 1 Of 14)

Figure 2-89 (5 Of 14) DDS Status Message Processor

Figure 2-89 (6 of 14).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The message indicating the DDSNAME, the primary DDNAME, and the backup DDNAME will also have the message OUT-OF-SERVICE.	DPP056I	DPPSMSGI
2	Ths message indicated in step 1 will be standard (indicating the backup is in service).	DPPS056I	DPPSMSGI

DPPSMSGI (Switch)

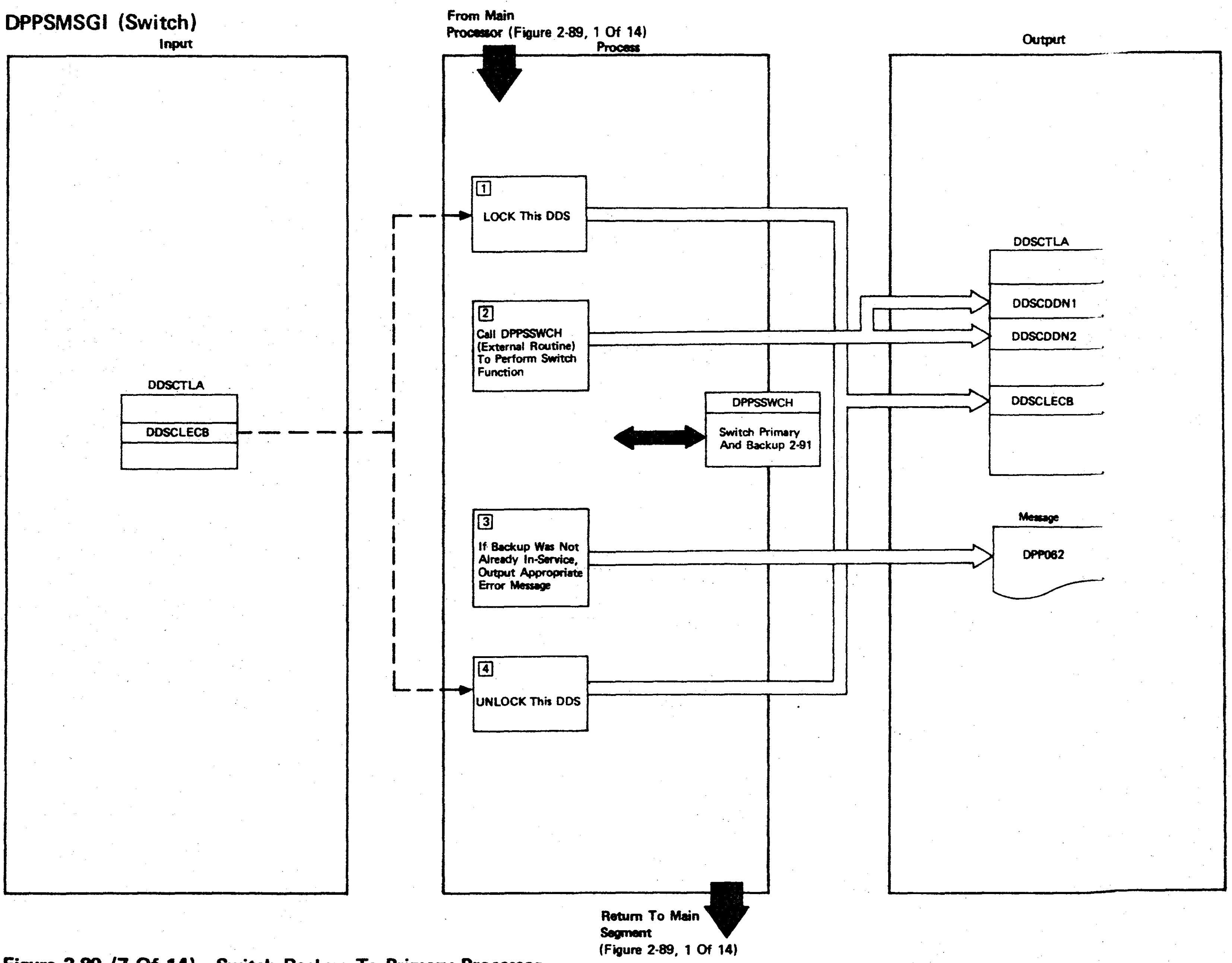
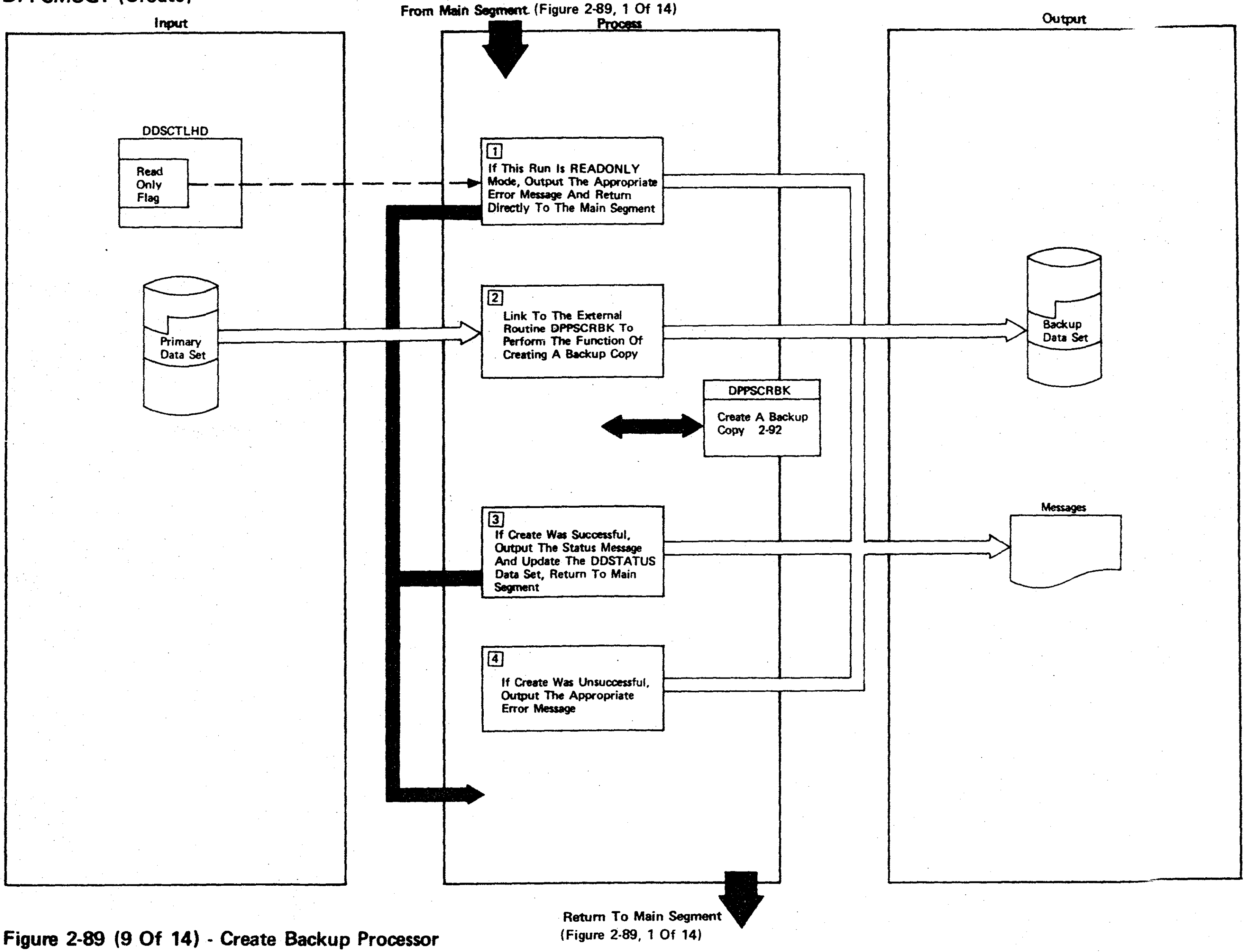

From Main Processor (Figure 2-89, 1 Of 14)
Input
Process
Output
DDSCTLA
DDSCLECB
1
LOCK This DDS
2
Call DPPSSWCH (External Routine) To Perform Switch Function
DPPSSWCH
Switch Primary And Backup 2-91
3
If Backup Was Not Already In-Service, Output Appropriate Error Message
4
UNLOCK This DDS
DDSCTLA
DDSCDDN1
DDSCDDN2
DDSCLECB
Message
DPP062
Return To Main Segment (Figure 2-89, 1 Of 14)

Figure 2-89 (7 Of 14) - Switch Backup To Primary Processor

Figure 2-89 (8 of 14).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The DDS LOCK function is invoked to prevent other tasks from interfering with the SWITCH operation.		DPPSMSGI
2	The internal routine DPPSSWCH will accomplish the switching of the DDNAMES and restart the I/O if necessary.		DPPSMSGI
3	Switching cannot be done if the backup was not already in service; if this is not true, output a message indicating that the switch could not be accomplished.	DPP062I	DPPSMSGI
4	The DDS LOCK function will be released to allow other tasks to continue processing this DDS.		DPPSMSGI

DPPSMSGI (Create)


From Main Segment (Figure 2-89, 1 Of 14)
Input
Process
Output
DDSCTLHD
Read Only Flag
Primary Data Set
DPPSCRBK
Create A Backup Copy 2-92
Backup Data Set
Messages
Return To Main Segment (Figure 2-89, 1 Of 14)

1 If This Run Is READONLY Mode, Output The Appropriate Error Message And Return Directly To The Main Segment

2 Link To The External Routine DPPSCRBK To Perform The Function Of Creating A Backup Copy

3 If Create Was Successful, Output The Status Message And Update The DDSTATUS Data Set, Return To Main Segment

4 If Create Was Unsuccessful, Output The Appropriate Error Message

Figure 2-89 (9 Of 14) - Create Backup Processor

Figure 2-89 (10 of 14).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	CREATE function is prohibited in the read only mode.	DPP889I	DPPSMSGI
2	The external module DPPSCRBK will direct the actual COPY operation and bring the backup in-service.		DPPSMSGI
3	The DDSTATUS data set contains the latest information concerning the DDS declarations. The status message indicates the CREATE completed successfully.	DPP056I	DPPSMSGI
4	Either the backup was already in service or the COPY operation was unsuccessful.	DPP064I DPP059I	DPPSMSGI

DPPSMSGI (Replace)

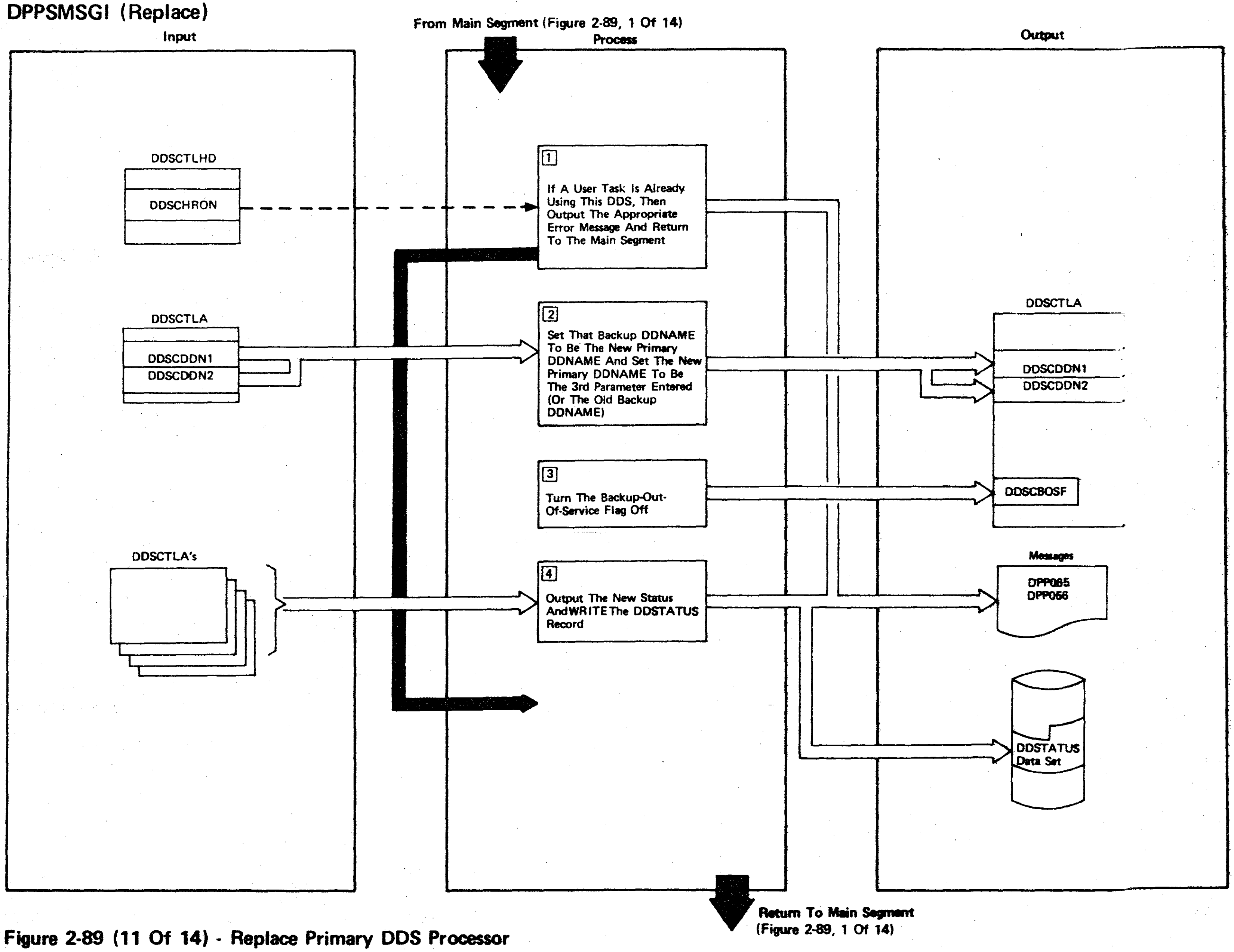

From Main Segment (Figure 2-89, 1 Of 14)
Input
Process
Output
DDSCTLHD
DDSCHRON
DDSCTLA
DDSCDDN1
DDSCDDN2
DDSCTLA's
1
If A User Task Is Already Using This DDS, Then Output The Appropriate Error Message And Return To The Main Segment
2
Set That Backup DDNAME To Be The New Primary DDNAME And Set The New Primary DDNAME To Be The 3rd Parameter Entered (Or The Old Backup DDNAME)
3
Turn The Backup-Out-Of-Service Flag Off
4
Output The New Status AndWRITE The DDSTATUS Record
DDSCTLA
DDSCDDN1
DDSCDDN2
DDSCBOSF
Messages
DPP065
DPP056
DDSTATUS Data Set
Return To Main Segment (Figure 2-89, 1 Of 14)

Figure 2-89 (11 Of 14) - Replace Primary DDS Processor

Figure 2-89 (12 of 14).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The REPLACE function is invalid if a DDSDCB is already opened against the DDS.	DPP065I	DPPSMSGI
2	If the user did not enter a third parameter, the default will be to change the old backup to the new primary. In either case, the old primary will become the new backup.		DPPSMSGI
3	The REPLACE function automatically sets the backup in service. (This is consistent since the old primary will be the new backup - the user can bring the backup out-of-service with a subsequent TAKE command.)		DPPSMSGI
4	The new status will be output and the current DDS declarations (as changed by the REPLACE request) will be recorded on the DDSTATUS data set.	DPP057I	DPPSMSGI

DPPSMSGI (Compare)

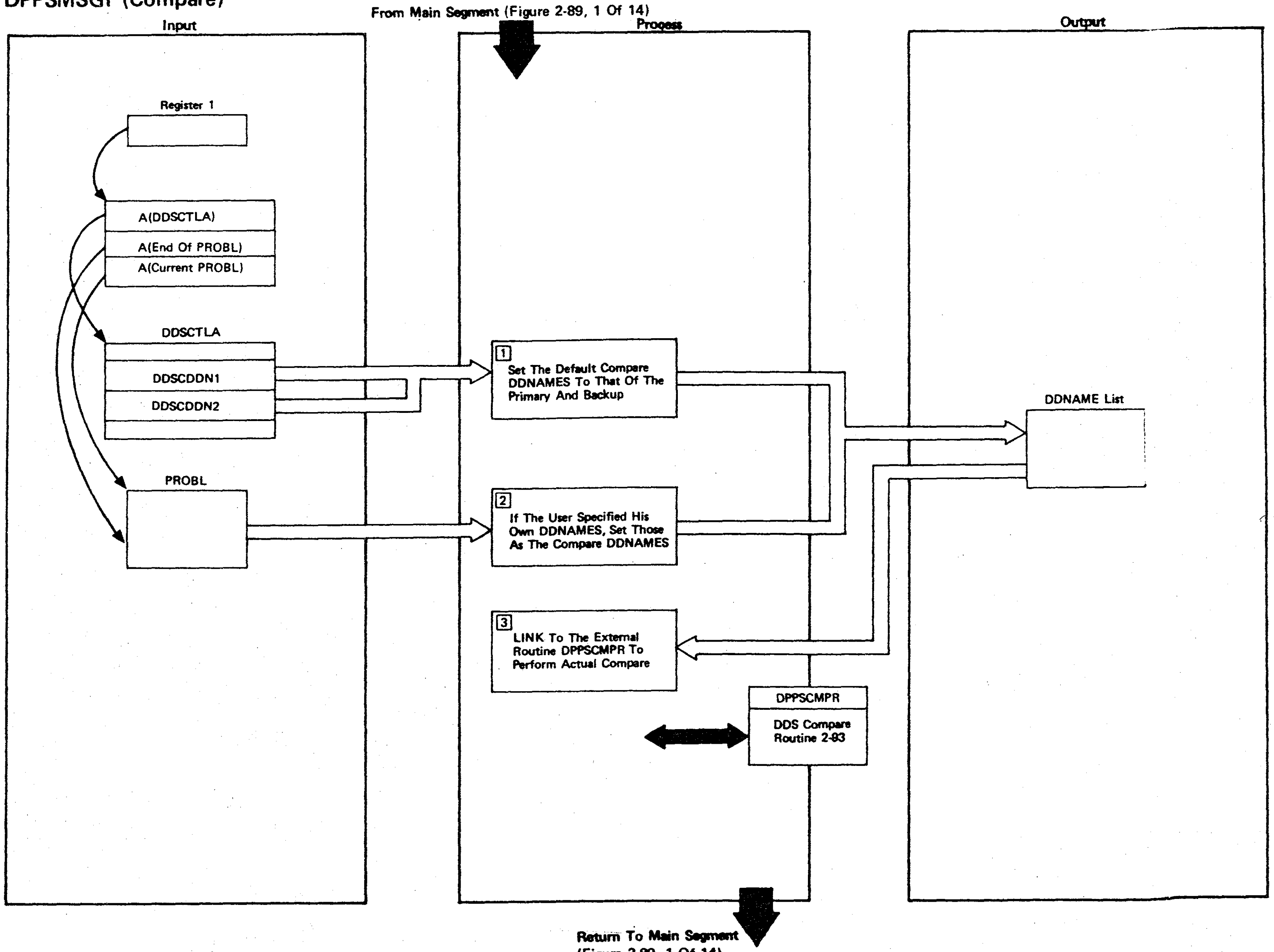
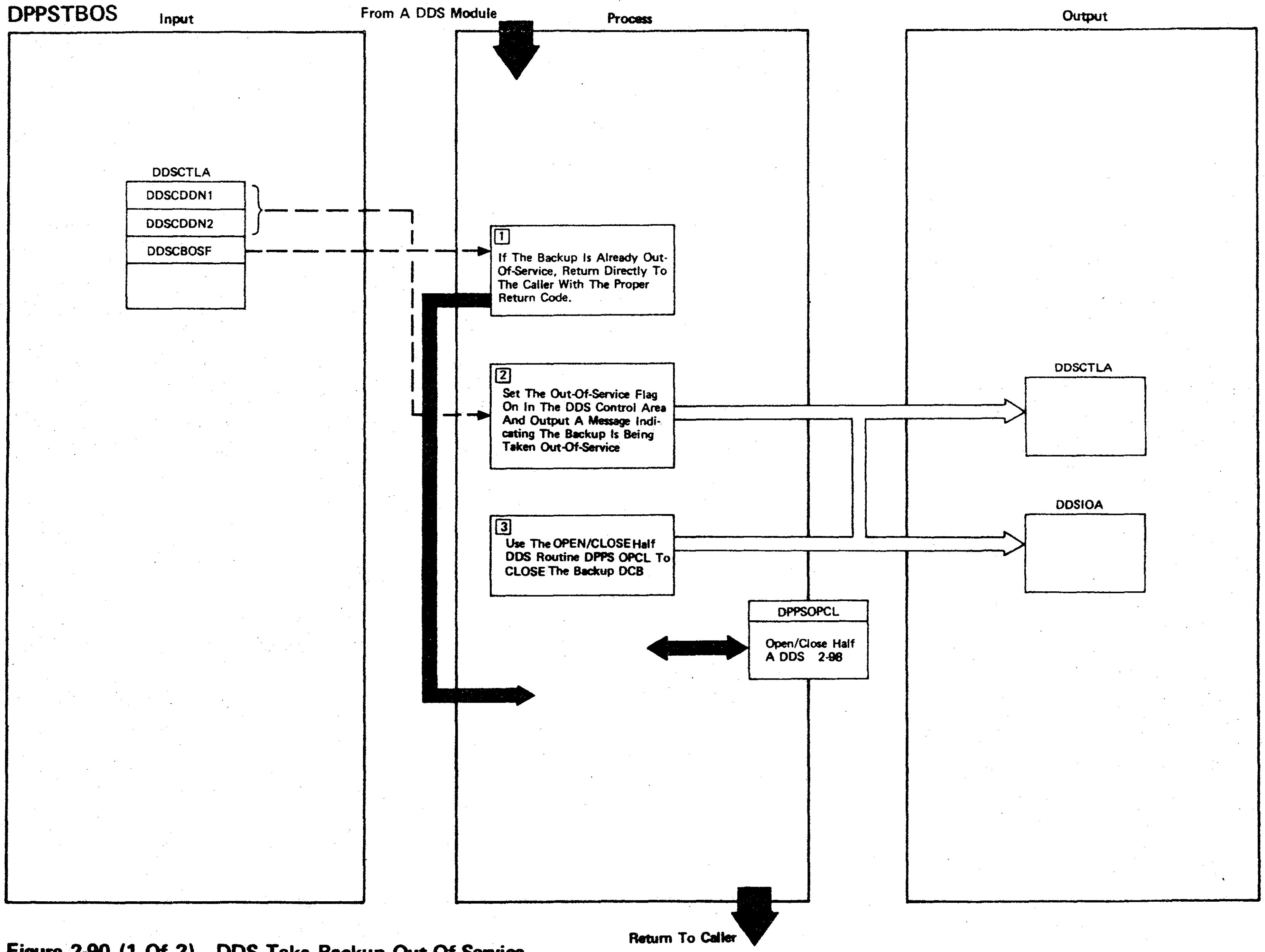

From Main Segment (Figure 2-89, 1 Of 14)
Input
Process
Output
Register 1
A(DDSCTLA)
A(End Of PROBL)
A(Current PROBL)
DDSCTLA
DDSCDDN1
DDSCDDN2
PROBL
1 Set The Default Compare DDNAMES To That Of The Primary And Backup
2 If The User Specified His Own DDNAMES, Set Those As The Compare DDNAMES
3 LINK To The External Routine DPPSCMPR To Perform Actual Compare
DPPSCMPR
DDS Compare Routine 2-83
DDNAME List
Return To Main Segment (Figure 2-89, 1 Of 14)

Figure 2-89 (13 Of 14) - DDS Compare Processor

Figure 2-89 (14 of 14).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The default for the COMPARE request is the compare to primary data set against the backup.		DPPSMSGI
2	The user can specify his own DDNAMES and effectively compare any two data sets of like DSORG.		DPPSMSGI
3	The external DDS routine DPPSCMPR will direct the actual compare and the messages.		DPPSMSGI



1 If The Backup Is Already Out-Of-Service, Return Directly To The Caller With The Proper Return Code.

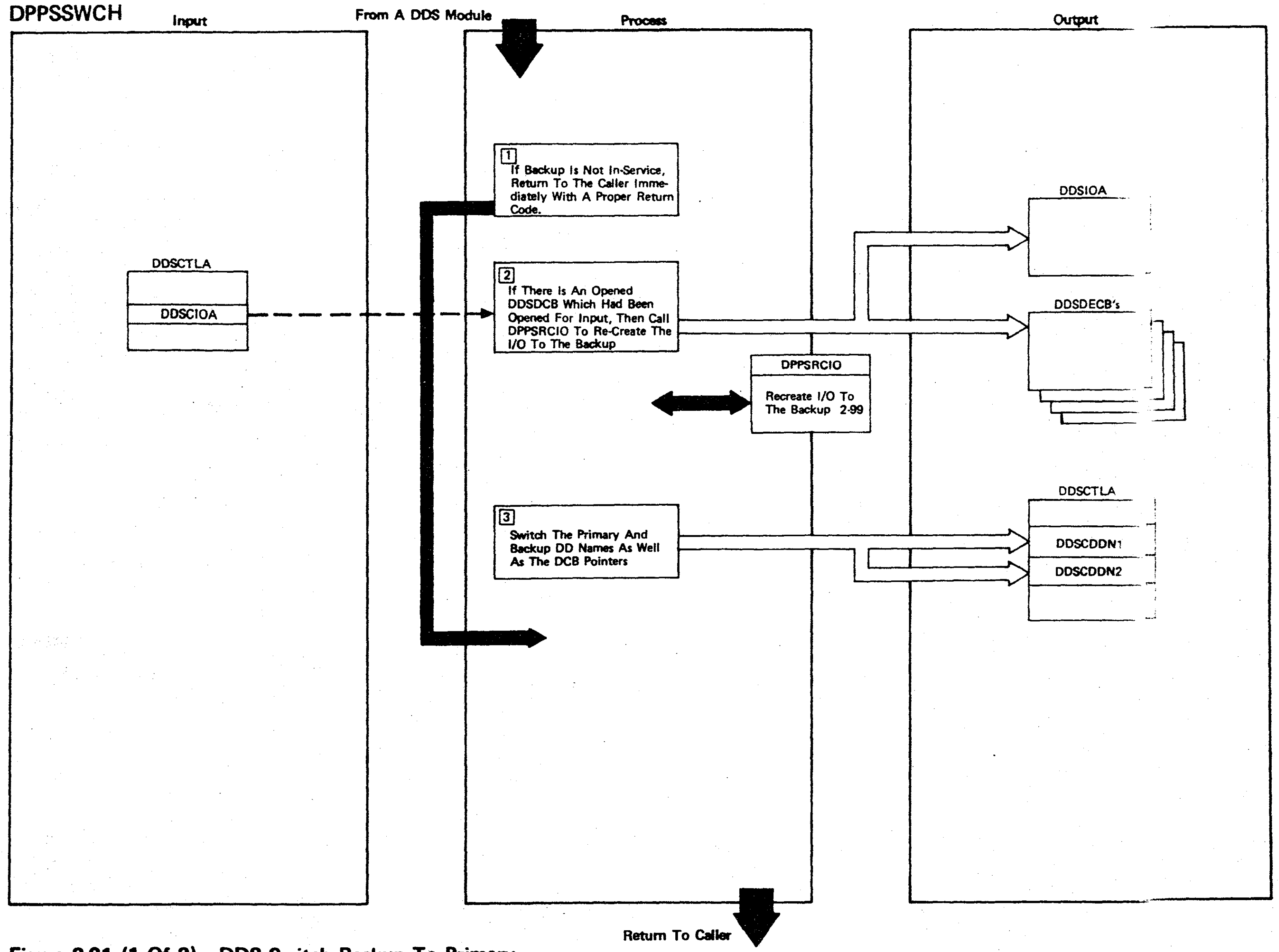
2 Set The Out-Of-Service Flag On In The DDS Control Area And Output A Message Indicating The Backup Is Being Taken Out-Of-Service

3 Use The OPEN/CLOSE Half DDS Routine DPPS OPCL To CLOSE The Backup DCB

Figure 2-90 (1 Of 2) - DDS Take Backup Out-Of-Service

Figure 2-90 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Another request may have taken the backup out-of-service before this module gains control.		DPPSTBOS
2	This message will indicate that the backup was taken out-of-service successfully, as requested.	DPP056I	DPPSTBOS
3	The backup DCB will be closed (asynchronously if opened by a different TCB) and the DDS status will be updated to reflect the change in serviceability of the backup.		DPPSTBOS



1 If Backup Is Not In-Service, Return To The Caller Immediately With A Proper Return Code.

2 If There Is An Opened DDSDCB Which Had Been Opened For Input, Then Call DPPSRCIO To Re-Create The I/O To The Backup

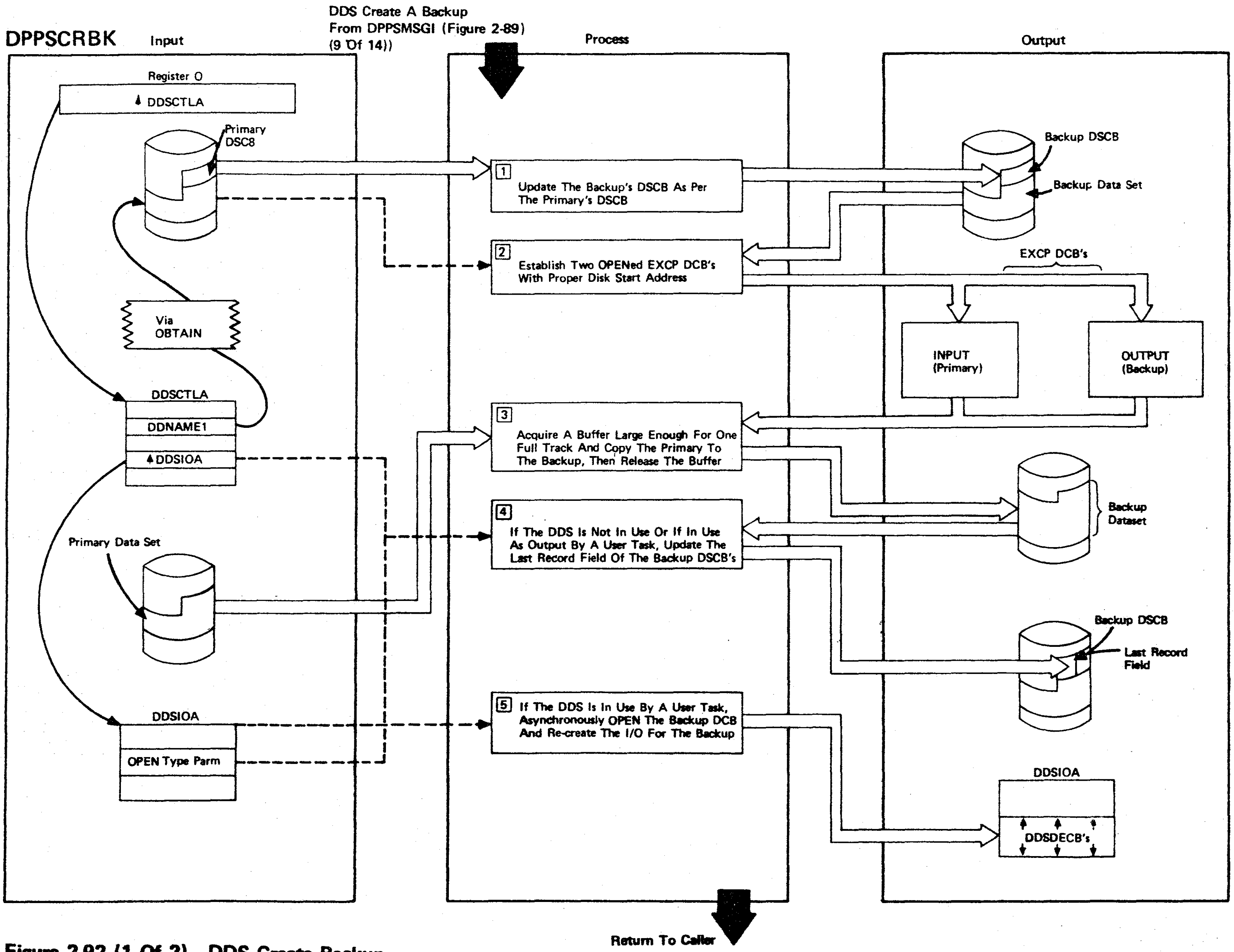
DPPSRCIO Recreate I/O To The Backup 2-99

3 Switch The Primary And Backup DD Names As Well As The DCB Pointers

Figure 2-91 (1 Of 2) - DDS Switch Backup To Primary

Figure 2-91 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Switching is not possible if the backup is not already in service.		DPPSSWCH
2	When a DDSDCB is opened for input, these I/O requests will not have been issued for the backup. This needs to be accomplished since the backup will become the primary data set.		DPPSSWCH
3	The DDNAME within the DDS control area indicates which is primary data set and which is backup for the CONTROL and COPY functions. The backup will be automatically taken out-of-service as part of the switch over function.		DPPSSWCH



1 Update The Backup's DSCB As Per The Primary's DSCB

2 Establish Two OPENed EXCP DCB's With Proper Disk Start Address

3 Acquire A Buffer Large Enough For One Full Track And Copy The Primary To The Backup, Then Release The Buffer

4 If The DDS Is Not In Use Or If In Use As Output By A User Task, Update The Last Record Field Of The Backup DSCB's

5 If The DDS Is In Use By A User Task, Asynchronously OPEN The Backup DCB And Re-create The I/O For The Backup

Figure 2-92 (1 Of 2) - DDS Create Backup

Figure 2-92 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The following fields will be updated in the backup DSCB: DSORG, RECF RECFM, OPTCD, KEYLEN, BLKSIZE, and LRECL.		DPPSCRBK
2	The primary EXCP will be opened for input, the backup EXCP will be opened for output, and the extents will be acquired from examining the Data Extent Block built by the OPEN.		DPPSCRBK
3	The buffer size will be found using the DEVTYP macro and the COPY operation will be track-to-track for every track in the extent.		DPPSCRBK
4	The last record field of the backup EXCP DCB (which would be pointing to the end of the extents as a result of the COPY operation) will be updated so that during the CLOSE the backup DSCB will be correctly updated.		DPPSCRBK
5	Each unchecked reserved DDSDECB for output or READ update will have to be restarted for the new backup.		DPPSCRBK

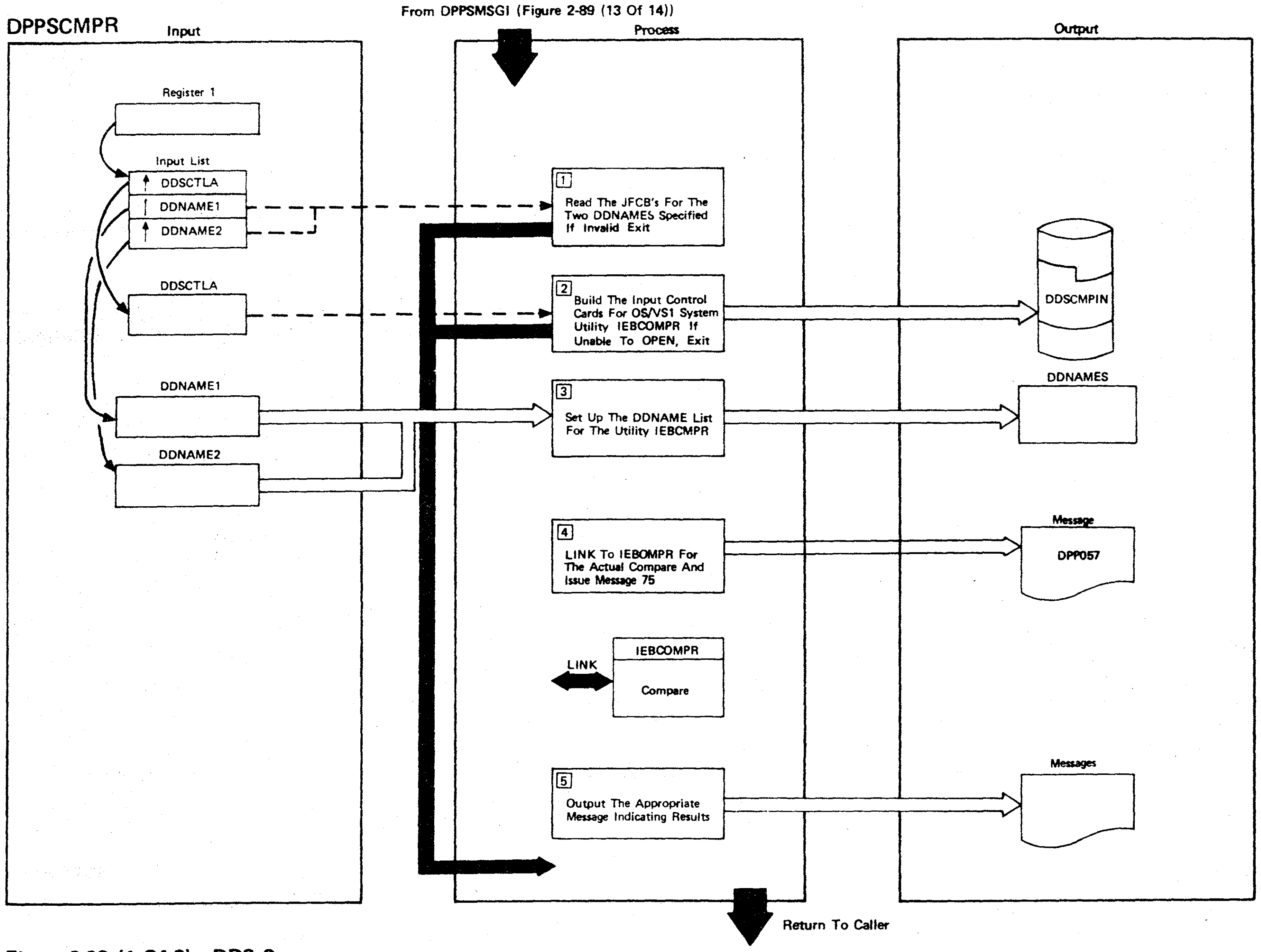

DPPSCMPR
Input
From DPPSMSGI (Figure 2-89 (13 Of 14))
Process
Output
Register 1
Input List
DDSCTLA
DDNAME1
DDNAME2
DDSCTLA
DDNAME1
DDNAME2
1
Read The JFCB's For The Two DDNAMES Specified If Invalid Exit
2
Build The Input Control Cards For OS/VS1 System Utility IEBCOMPR If Unable To OPEN, Exit
3
Set Up The DDNAME List For The Utility IEBCMPR
4
LINK To IEBOMPR For The Actual Compare And Issue Message 75
LINK
IEBCOMPR
Compare
5
Output The Appropriate Message Indicating Results
DDSCMPIN
DDNAMES
Message
DPP057
Messages
Return To Caller

Figure 2-93 (1 Of 2) - DDS Compare

Figure 2-93 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If either of the JFCBs could not be read, output the message UNABLE TO READ JFCBs and return to DPPSMSGI.	DPP073I	DPPSCMPR
2	If the DSORGs are not the same type, output message NOT SAME TYPE and return to DPPSMSGI.	DPP074I	DPPSCMPR
	If the control card data set DDSCMPIN cannot be opened and written successfully, output message DDS COMPARE CONTROL CARD ERROR and return to DPPSMSGI.	DPP078I	DPPSCMPR
3	The two compare DDNAMES should be supplied by DPPSMSGI, DDSCMPIN will replace SYSIN, and COMPRINT replaces SYSPRINT.		DPPSCMPR
4	The DDS will be locked during the execution of IEBCOMPR	DPP075I	DPPSCMPR
5	One of the following three messages will result:		DPPSCMPR
	COMPARE ENDED, DATA SETS ARE EQUAL	DPP036I	DPPSCMPR
	COMPARE ENDED, DATA SETS ARE UNEQUAL	DPP076I	
	COMPARE RESULTS ARE ON COMPRINT.	DPP077I	

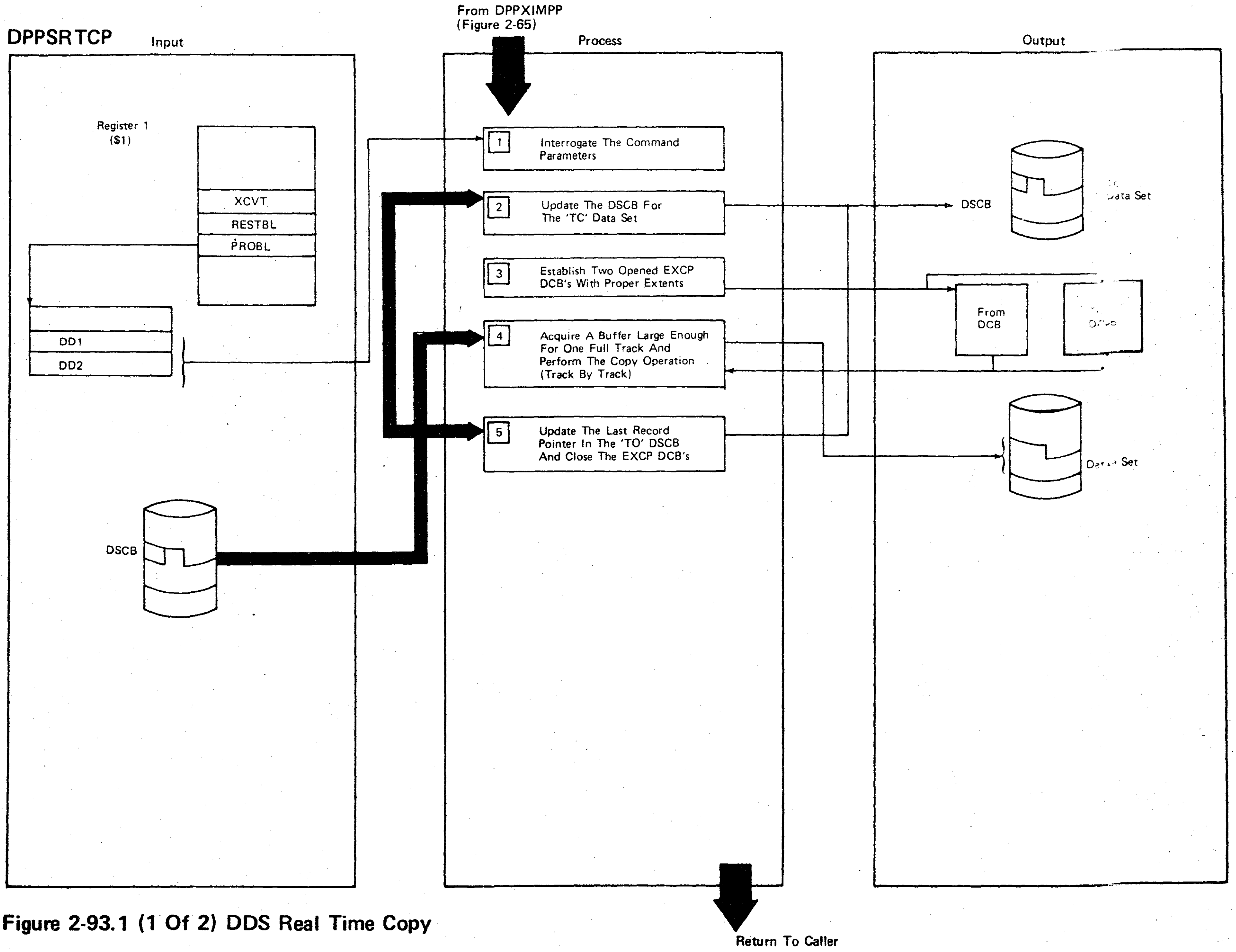

DPPSRTCP
Input
Process
Output
From DPPXIMPP
(Figure 2-65)
Register 1
($1)
XCVT
RESTBL
PROBL
DD1
DD2
DSCB
1 Interrogate The Command Parameters
2 Update The DSCB For The 'TC' Data Set
3 Establish Two Opened EXCP DCB's With Proper Extents
4 Acquire A Buffer Large Enough For One Full Track And Perform The Copy Operation (Track By Track)
5 Update The Last Record Pointer In The 'TO' DSCB And Close The EXCP DCB's
DSCB
To Data Set
From DCB
To DCB
Data Set
Return To Caller

Figure 2-93.1 (1 Of 2) DDS Real Time Copy

Figure 2-93.1 (2 Of 2)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The DDNAMEs of the 'from' and 'to' data sets should be in the TIOT.	865 866 867	DPPSRTCP
2	The following fields will be updated: DSORG, RECFM, OPTCD, KEYLEN, BLKSIZE, and LRECL .	868 869 870 871	DPPSRTCP
3	The 'from' EXCP DCB will be opened for input, the 'to' for output, and the extents will be obtained from the DEBs.		DPPSRTCP
4	The buffer size will be determined using the DEVTYP macro and the copy operation will be track-to-track.		DPPSRTCP
5	The lost record field of the 'to' DSCB will be set according to that of the 'from' DCB.		DDSRTCP

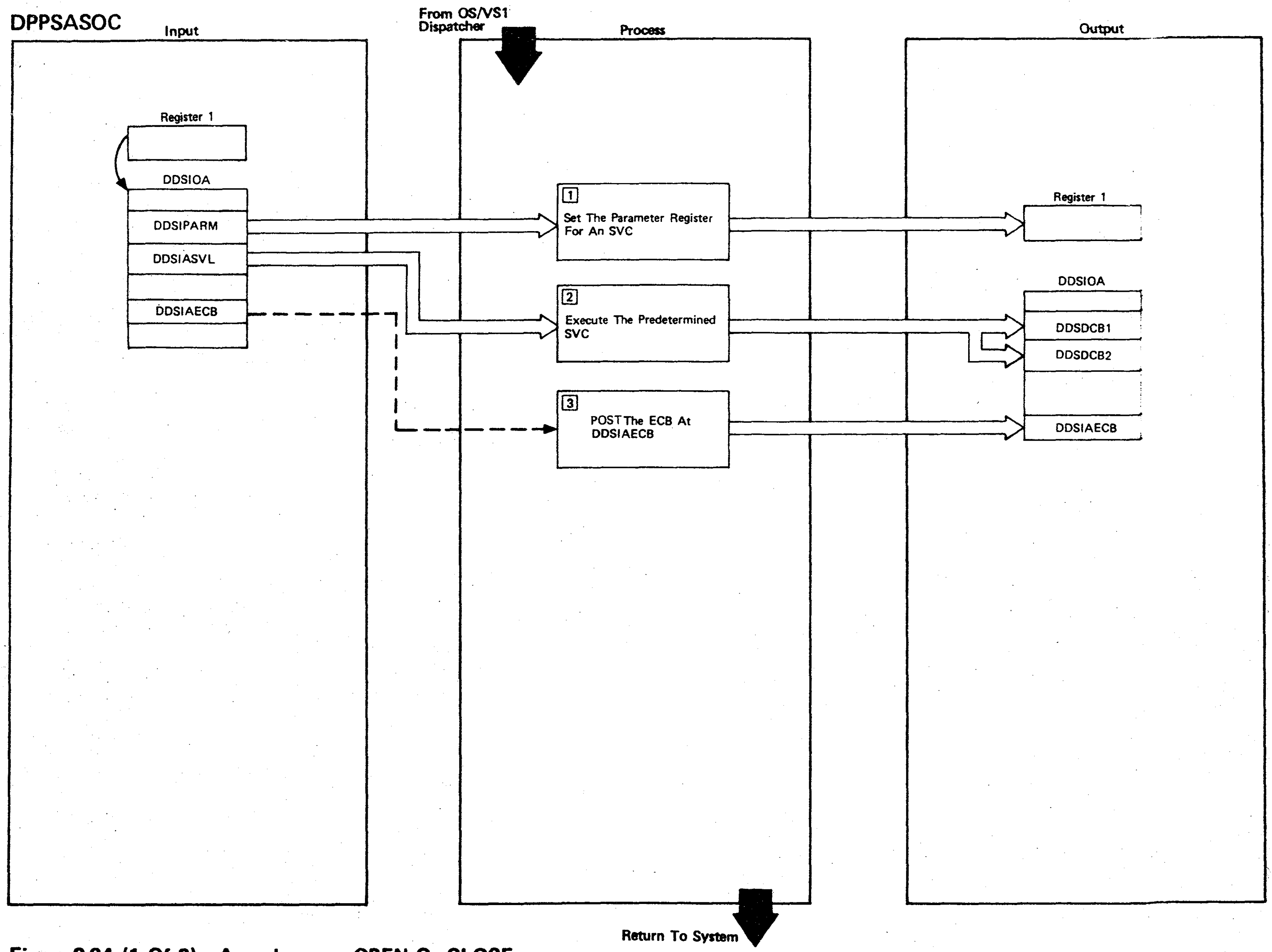
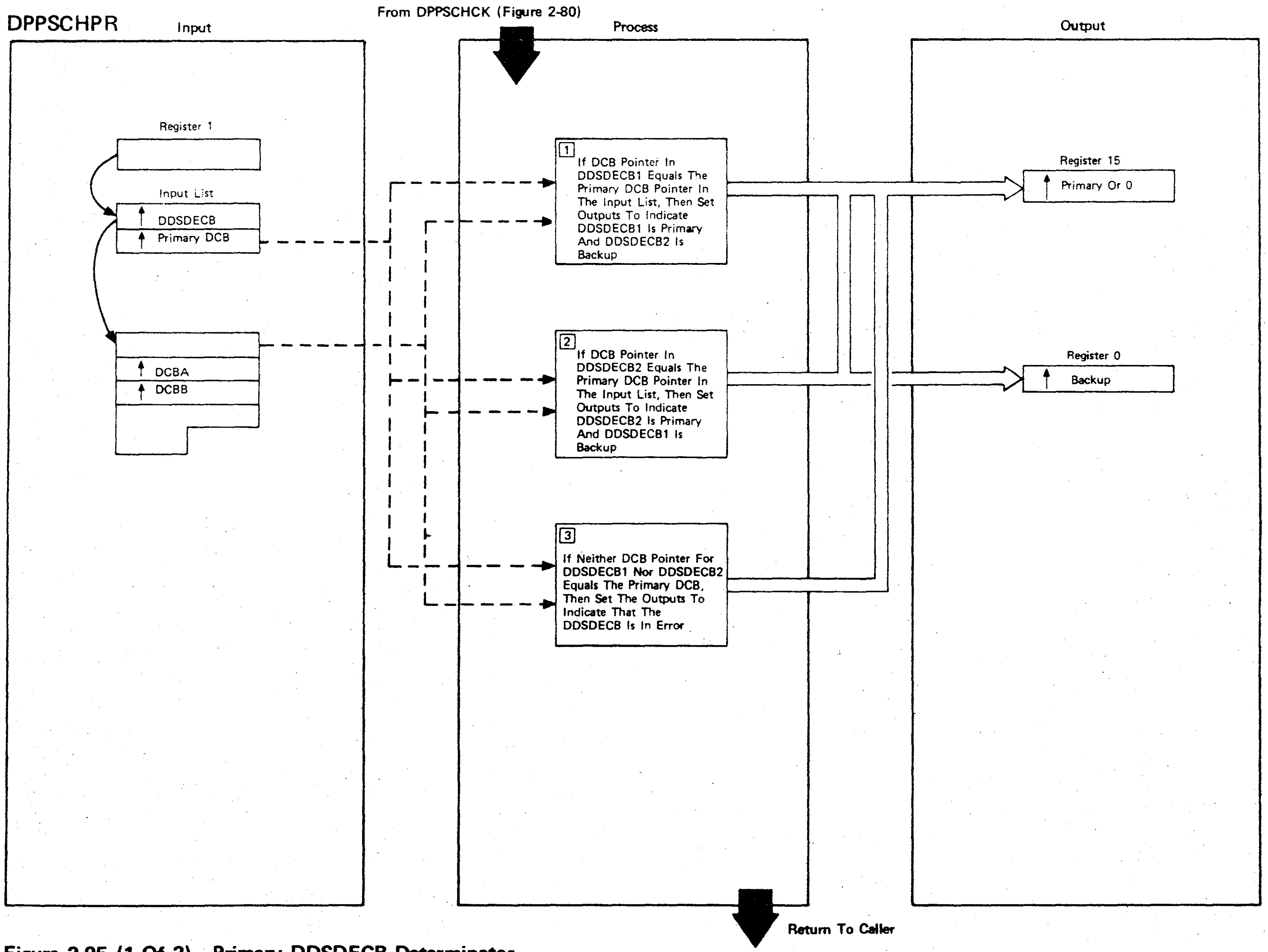


Figure 2-94 (1 Of 2) - Asynchronous OPEN Or CLOSE

Figure 2-94 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The parameters DDSIPARM and DDSIASVC are set by DPPSOPCL which builds an IQE and causes this routine to be dispatched via a related IRB. The task under which the IRB is running should be the same as the task that opened this DDS (this determination is made by DPPSOPCL).		DPPSASOC
2	The particular request (OPEN or CLOSE) will have been set already by the module that had called DPPSOPCL. The SVC instruction code will be in the input parameter DDSIASVC. Either DDSDCB1 or DDSDCB2 will be opened or closed, depending on the inputs.		DPPSASOC
3	The ECB at DDSIAECB is the one the task for DPPSOPCL is waiting on. This results in a RETURN TO CALLER through the IRB path.		DPPSASOC



1 If DCB Pointer In DDSDECB1 Equals The Primary DCB Pointer In The Input List, Then Set Outputs To Indicate DDSDECB1 Is Primary And DDSDECB2 Is Backup

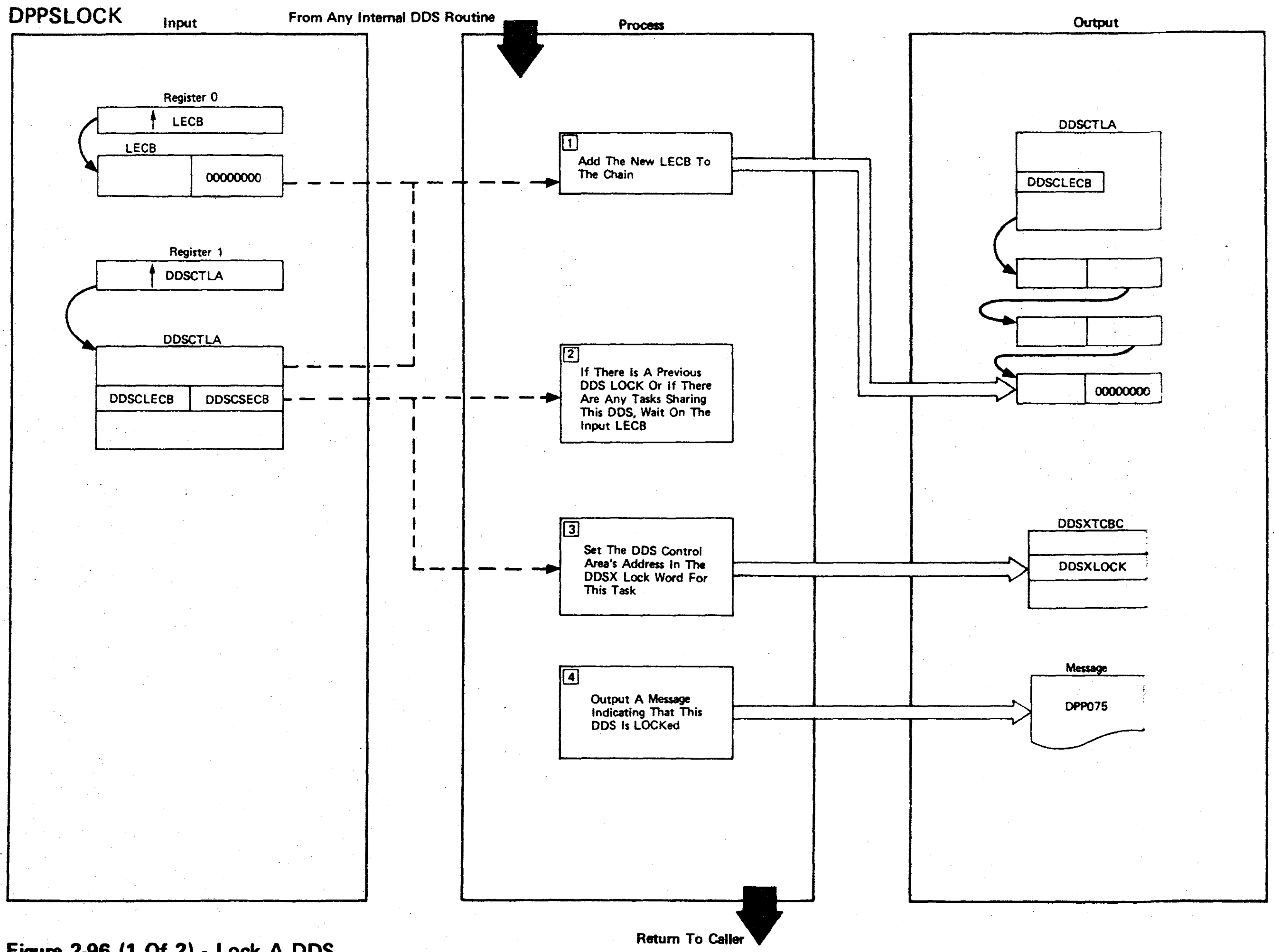
2 If DCB Pointer In DDSDECB2 Equals The Primary DCB Pointer In The Input List, Then Set Outputs To Indicate DDSDECB2 Is Primary And DDSDECB1 Is Backup

3 If Neither DCB Pointer For DDSDECB1 Nor DDSDECB2 Equals The Primary DCB, Then Set The Outputs To Indicate That The DDSDECB Is In Error

Figure 2-95 (1 Of 2) - Primary DDSDECB Determinator

Figure 2-95 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	This condition indicates that no switching has occurred since the execution of the READ/WRITE operation.		DPPSCHPR
2	This condition indicates that a switchover has occurred and that the old backup DDSDECB is the new primary DDSDECB.		DPPSCHPR
3	This condition indicates a logic error pertaining to this DDSDECB.		DPPSCHPR



1 Add The New LECB To The Chain

2 If There Is A Previous DDS LOCK Or If There Are Any Tasks Sharing This DDS, Wait On The Input LECB

3 Set The DDS Control Area's Address In The DDSX Lock Word For This Task

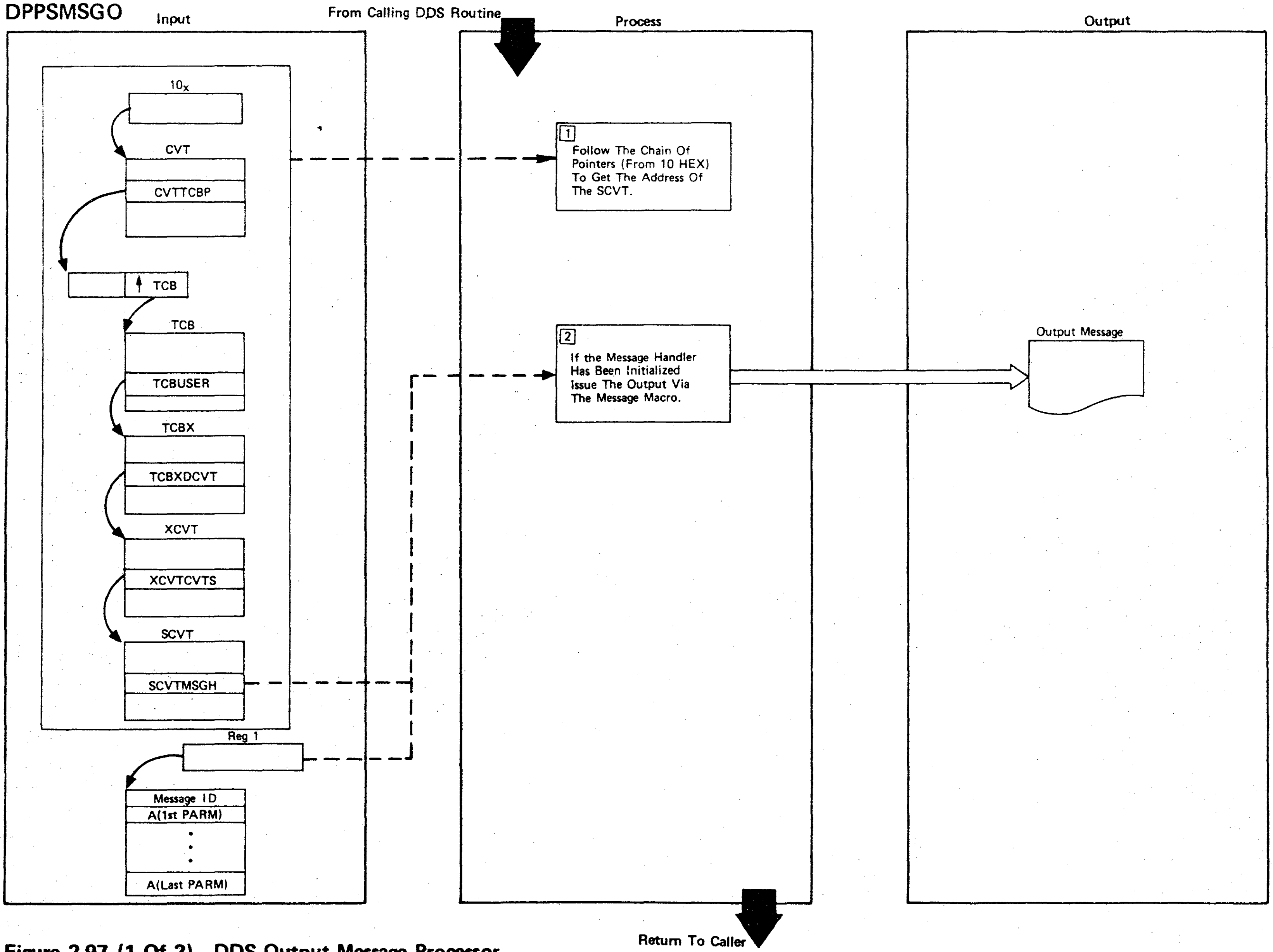
4 Output A Message Indicating That This DDS Is LOCKed

Figure 2-96 (1 Of 2) - Lock A DDS

Figure 2-96 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The Special Real Time Operating System LOCK (defined by DPPSINIT) will be used while adding this LECB to the chain.		DPPSLOCK
2	The LECB will be posted when all previous DDS LOCKS are released and all current users unshare this DDS.		DPPSLOCK
3	The DDSX task chain is used to indicate that this DDS is locked out in case this task should be prematurely terminated.		DPPSLOCK
4	The message DDSNAME IS LOCKED will be output to inform the operator of the LOCK condition.	DPP057I	DPPSLOCK

DPPSMSGO



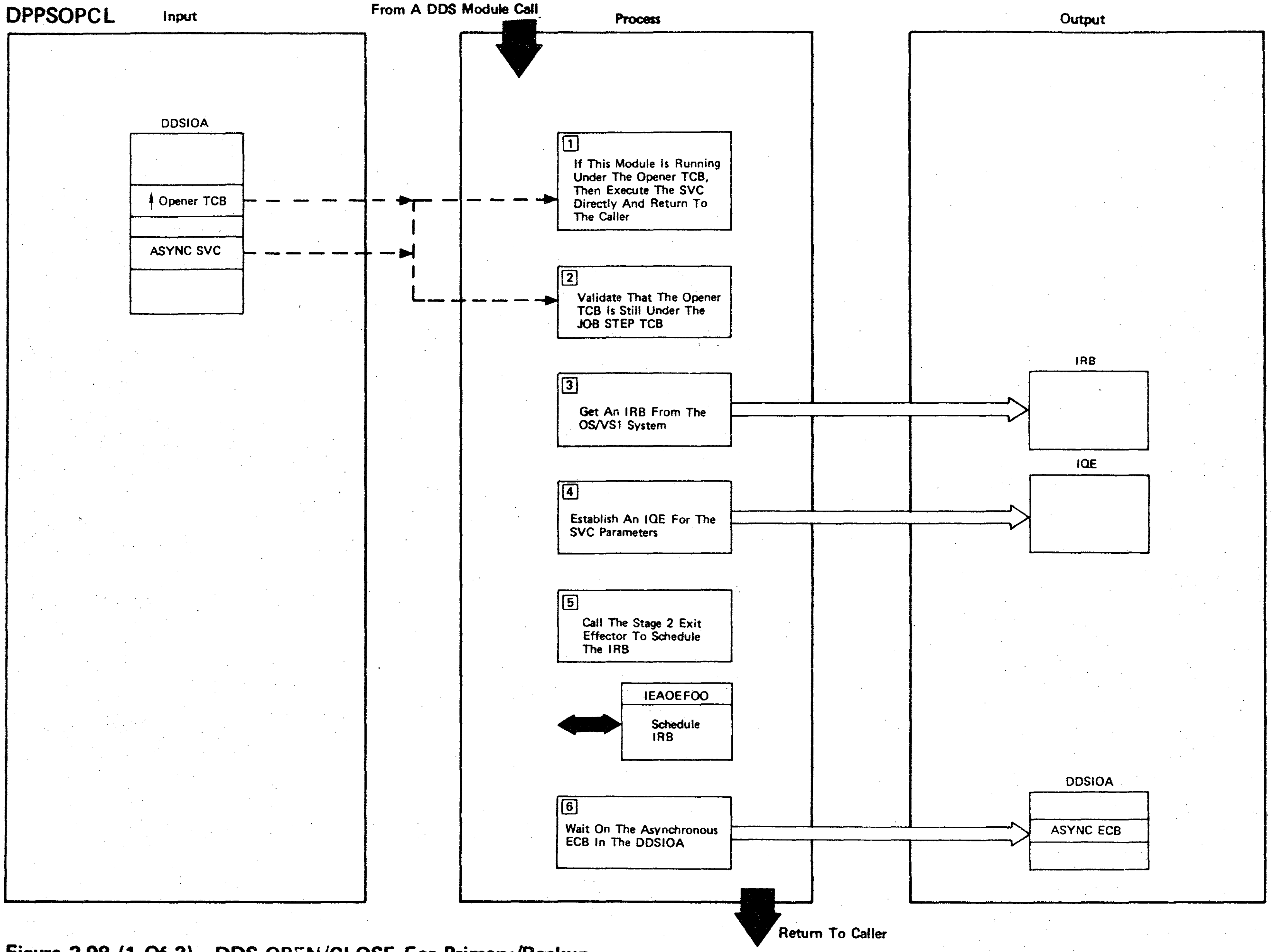
1 Follow The Chain Of Pointers (From 10 HEX) To Get The Address Of The SCVT.

2 If the Message Handler Has Been Initialized Issue The Output Via The Message Macro.

Figure 2-97 (1 Of 2) - DDS Output Message Processor

Figure 2-97 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	This chain will give the address of the SCVT from which it can be determined whether or not the Special Real Time Operating System message handler has been initialized.		DPPSMSGO
2	The number of variables in the message has been predefined with the offline utility DPPXUTIL. This message macro passes the maximum number of variables for DDS (5).		DPPSMSGO



1 If This Module Is Running Under The Opener TCB, Then Execute The SVC Directly And Return To The Caller

2 Validate That The Opener TCB Is Still Under The JOB STEP TCB

3 Get An IRB From The OS/VS1 System

4 Establish An IQE For The SVC Parameters

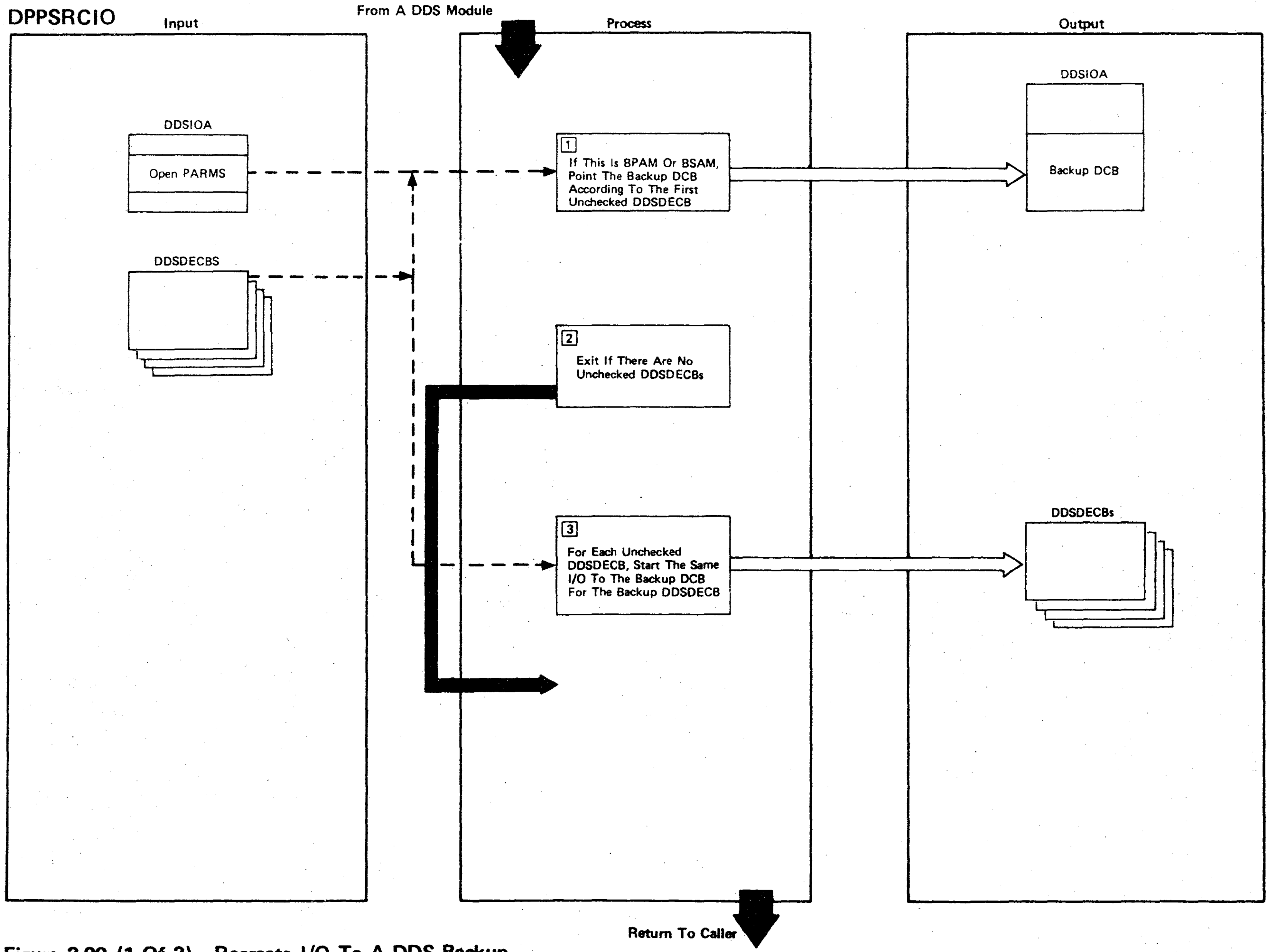
5 Call The Stage 2 Exit Effector To Schedule The IRB

6 Wait On The Asynchronous ECB In The DDSIOA

Figure 2-98 (1 Of 2) - DDS OPEN/CLOSE For Primary/Backup

Figure 2-98 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The SVC to be executed and the address of the task requesting the service are both in the DDSIOA input.		DPPSOPCL
2	The validation algorithm assumes that the priority of the task requesting the service is less than or equal to the priority of the job step TCB.		DPPSOPCL
3	The IRB is obtained using the CIRB macro.		DPPSOPCL
4	The IQE contains the parameters for the subsequent routine DPPSASOC which will be entered asynchronously.		DPPSOPCL
5	The stage 2 exit schedules the IQE in step 4 to be executed at the next task switch.		DPPSOPCL
6	The WAIT causes a task switch yielding control to DPPSASOC under the IRB of step three. DPPSASOC posts the ECB in the DDSIOA when completed.		DPPSOPCL



1 If This Is BPAM Or BSAM, Point The Backup DCB According To The First Unchecked DDSDECB

2 Exit If There Are No Unchecked DDSDECBs

3 For Each Unchecked DDSDECB, Start The Same I/O To The Backup DCB For The Backup DDSDECB

Figure 2-99 (1 Of 2) - Recreate I/O To A DDS Backup

Figure 2-99 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The OPEN parameters determine if POINT is allowed. The correct TTR can be obtained using the OS/VS1 conversion routine for CCHHR to TTR against the primary DDSDECB IOB.		DPPSRCIO
2	Unchecked DDSDECB require no I/O restart.		DPPSRCIO
3	Each unchecked DDSDECB is chained sequentially according to the time that the I/O was issued, so the same order can be used for the backup DDSDECB.		DPPSRCIO

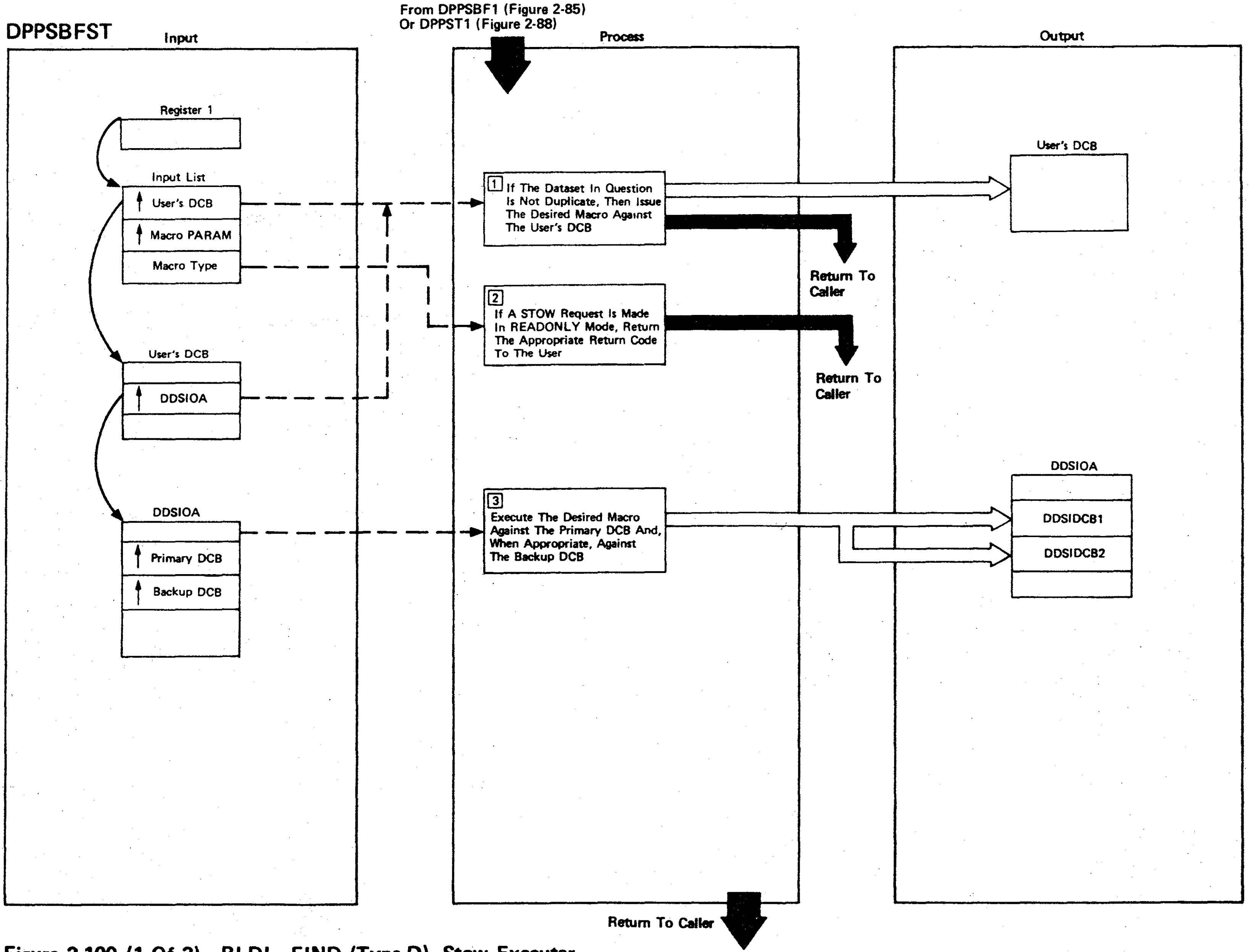

DPPSBFST
From DPPSBF1 (Figure 2-85)
Or DPPST1 (Figure 2-88)
Input
Process
Output
Register 1
Input List
User's DCB
Macro PARAM
Macro Type
User's DCB
DDSIOA
DDSIOA
Primary DCB
Backup DCB
1 If The Dataset In Question Is Not Duplicate, Then Issue The Desired Macro Against The User's DCB
2 If A STOW Request Is Made In READONLY Mode, Return The Appropriate Return Code To The User
3 Execute The Desired Macro Against The Primary DCB And, When Appropriate, Against The Backup DCB
Return To Caller
Return To Caller
User's DCB
DDSIOA
DDSIDCB1
DDSIDCB2
Return To Caller

Figure 2-100 (1 Of 2) - BLDL, FIND (Type-D), Stow Executer

Figure 2-100 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The chain of DDS control areas is searched for one whose DDSIOA pointer matches the input (user's DCB DDSIOA pointer). Finding a match signifies that the data set is a DDS.		DPPSBFST
2	No output is allowed in read only mode (backup computer).		DPPSBFST
3	The DDS is shared during the executions of the macro. If a CONTROL function is needed (switchover if an error occurs on the primary, or take backup out-of-service if an error occurs on the backup of output), an interim DDSLOCK is placed on the DDS until the CONTROL function is completed.		DPPSBFST

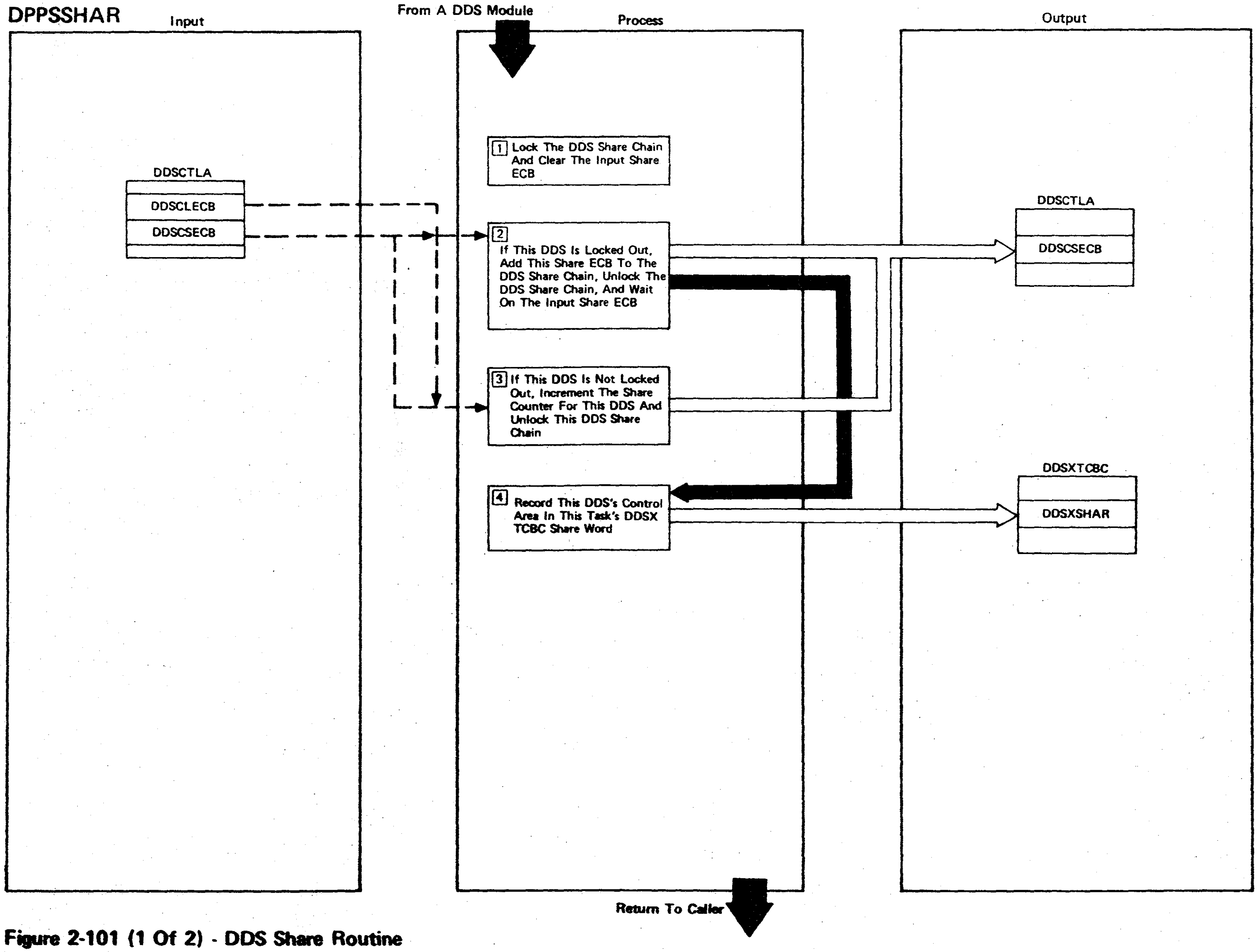

DPPSSHAR
Input
From A DDS Module
Process
Output
DDSCTLA
DDSCLECB
DDSCSECB
1 Lock The DDS Share Chain And Clear The Input Share ECB
2 If This DDS Is Locked Out, Add This Share ECB To The DDS Share Chain, Unlock The DDS Share Chain, And Wait On The Input Share ECB
3 If This DDS Is Not Locked Out, Increment The Share Counter For This DDS And Unlock This DDS Share Chain
4 Record This DDS's Control Area In This Task's DDSX TCBC Share Word
DDSCTLA
DDSCSECB
DDSXTCBC
DDSXSHAR
Return To Caller

Figure 2-101 (1 Of 2) - DDS Share Routine

Figure 2-101 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The share chain must be locked while additions or modifications are being made to it.		DPPSSHAR
2	The LOCKING task will post all share ECBs waiting to share when it releases the DDS LOCK.		DPPSSHAR
3	When this task unshares the DDS the share count will be decremented.		DPPSSHAR
4	If this task should ABEND, the DDS cleanup routine could unshare this DDS by examining the DDSXTCBC chain.		DPPSSHAR

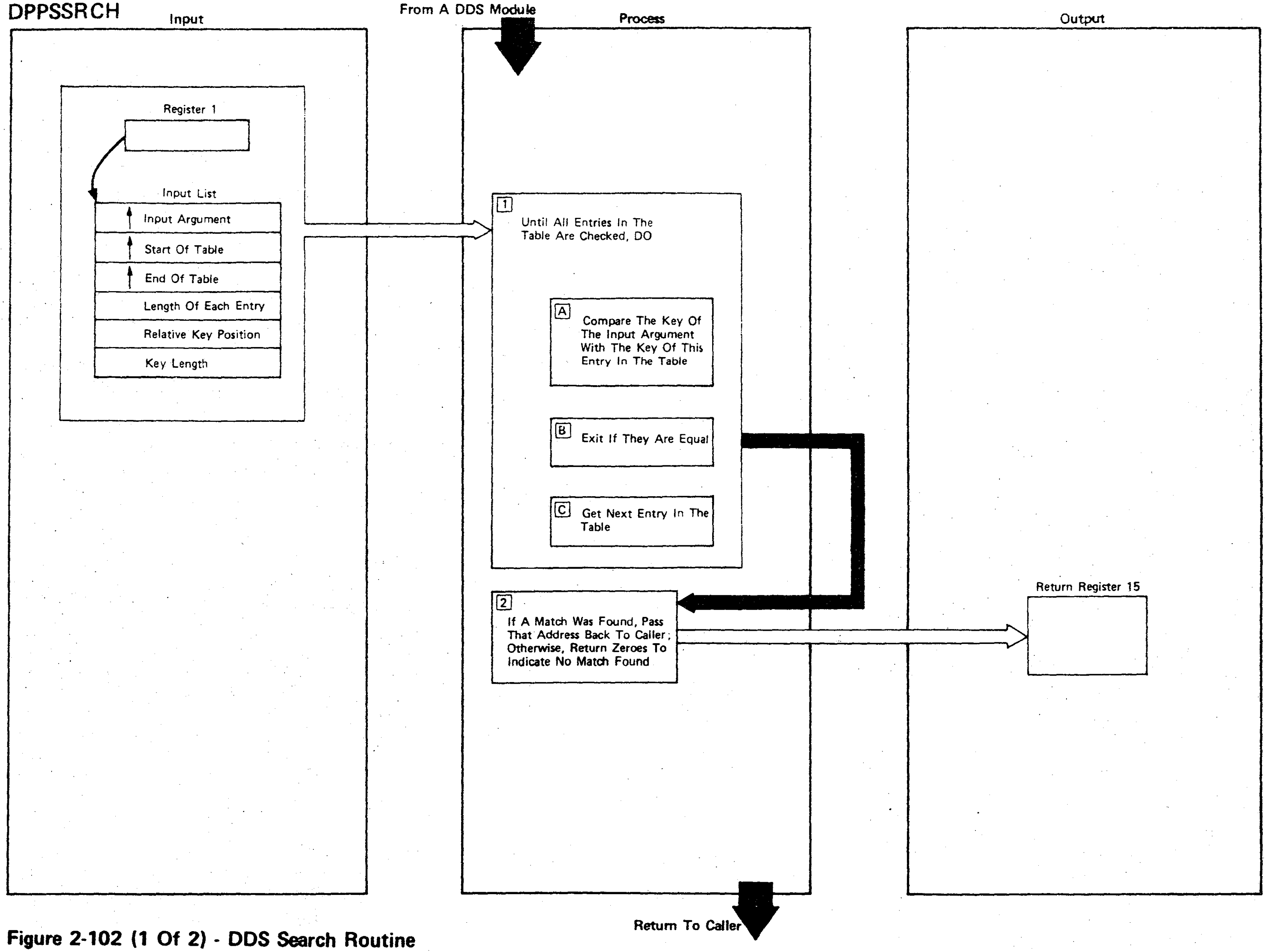

DPPSSRCH
Input
From A DDS Module
Process
Output
Register 1
Input List
Input Argument
Start Of Table
End Of Table
Length Of Each Entry
Relative Key Position
Key Length
1 Until All Entries In The Table Are Checked, DO
A Compare The Key Of The Input Argument With The Key Of This Entry In The Table
B Exit If They Are Equal
C Get Next Entry In The Table
2 If A Match Was Found, Pass That Address Back To Caller; Otherwise, Return Zeroes To Indicate No Match Found
Return Register 15
Return To Caller

Figure 2-102 (1 Of 2) - DDS Search Routine

Figure 2-102 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Each entry in the table should be checked, starting with the first and proceeding serially until the last entry.		DPPSSRCH
2	The caller will be notified by the contents of return register 15 if a match was found. Zero indicates that the argument has no match in the table, while a nonzero value will be the address of the entry matching the input argument.		DPPSSRCH

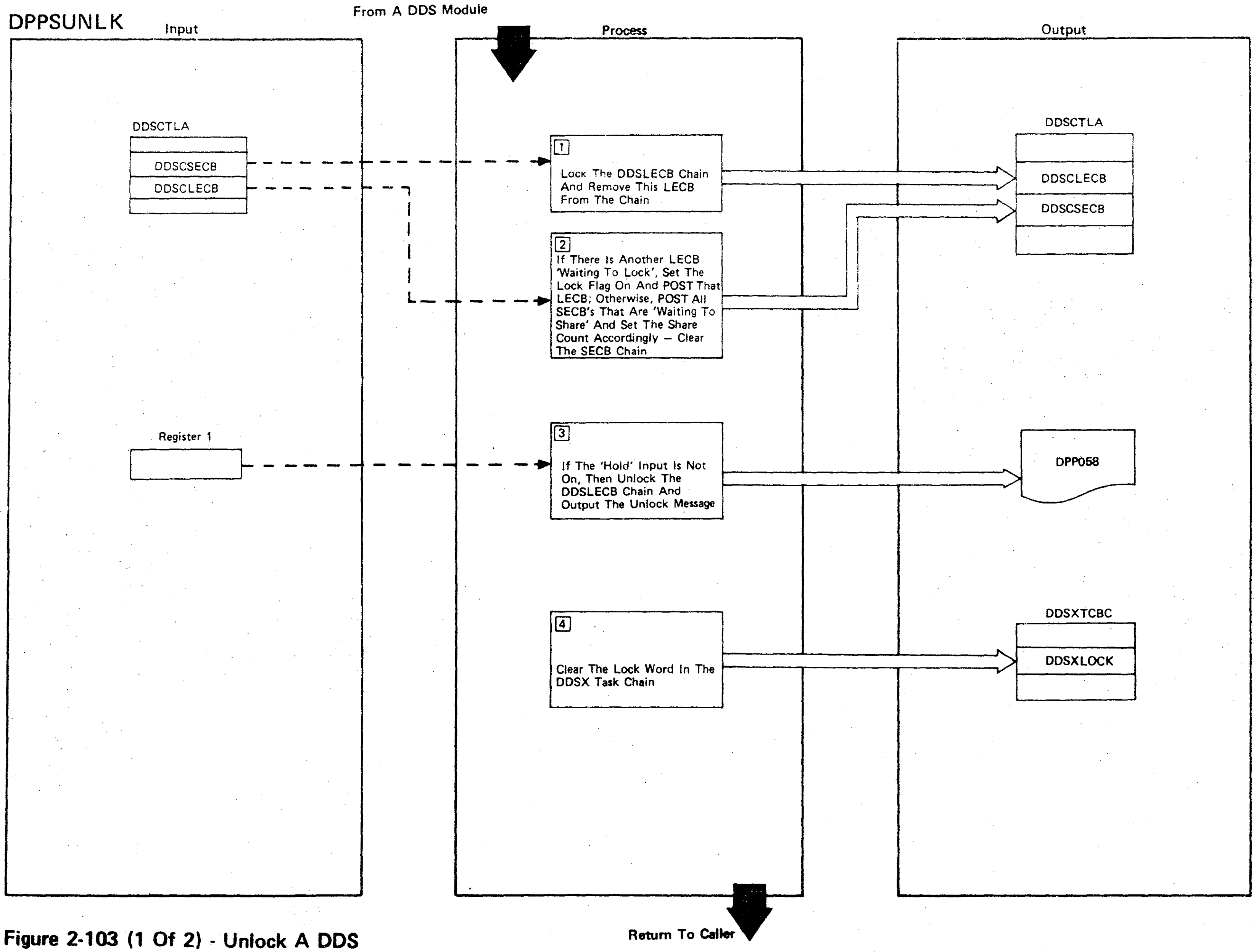
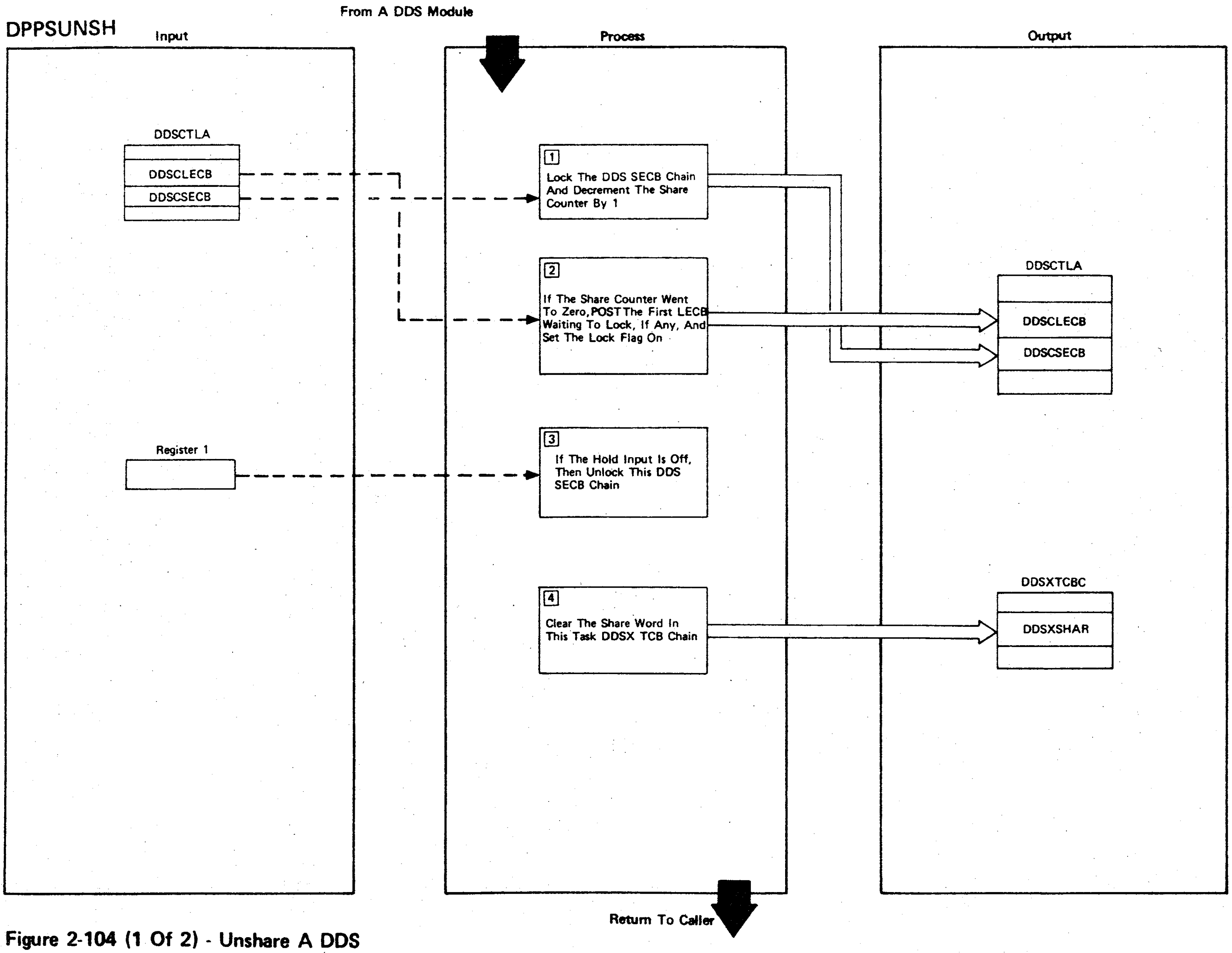

DPPSUNLK
Input
From A DDS Module
Process
Output
DDSCTLA
DDSCSECB
DDSCLECB
1
Lock The DDSLECB Chain And Remove This LECB From The Chain
2
If There Is Another LECB 'Waiting To Lock', Set The Lock Flag On And POST That LECB; Otherwise, POST All SECB's That Are 'Waiting To Share' And Set The Share Count Accordingly — Clear The SECB Chain
DDSCTLA
DDSCLECB
DDSCSECB
Register 1
3
If The 'Hold' Input Is Not On, Then Unlock The DDSLECB Chain And Output The Unlock Message
DPP058
4
Clear The Lock Word In The DDSX Task Chain
DDSXTCBC
DDSXLOCK
Return To Caller

Figure 2-103 (1 Of 2) - Unlock A DDS

Figure 2-103 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Use the Special Real Time Operating System LOCK to inhibit other tasks from accessing the DDSLECB chain while being modified by this task.		DPPSUNLK
2	A task that is waiting to LOCK a DDS takes precedence over all tasks waiting to share that same DDS.		DPPSUNLK
3	The caller has the option of holding the DDSLECB chain in LOCK (by setting the high-order bit of register 1 on) so he may use other DDS modules to make further modification to the chain. The message indicates that the DDS LOCK has been released.	DPP058I	DPPSUNLK
4	The UNLOCK function should erase the LOCK pointer in this task's DDSX chain.		DPPSUNLK

DPPSUNSH



1 Lock The DDS SECB Chain And Decrement The Share Counter By 1

2 If The Share Counter Went To Zero,POST The First LECB Waiting To Lock, If Any, And Set The Lock Flag On

3 If The Hold Input Is Off, Then Unlock This DDS SECB Chain

4 Clear The Share Word In This Task DDSX TCB Chain

Figure 2-104 (1 Of 2) - Unshare A DDS

Figure 2-104 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Use the Special Real Time Operating System LOCK to prevent other tasks from using the DDSSECB chain while it's being modified.		DPPSUNSH
2	Only after all current users are finished with a DDS can it be locked.		DPPSUNSH
3	The caller could have specified HOLD (setting the high-order bit of register 1 on) for further modifications.		DPPSUNSH
4	The share pointer to this DDS control area in this task's DDSX chain should be zeroed.		DPPSUNSH

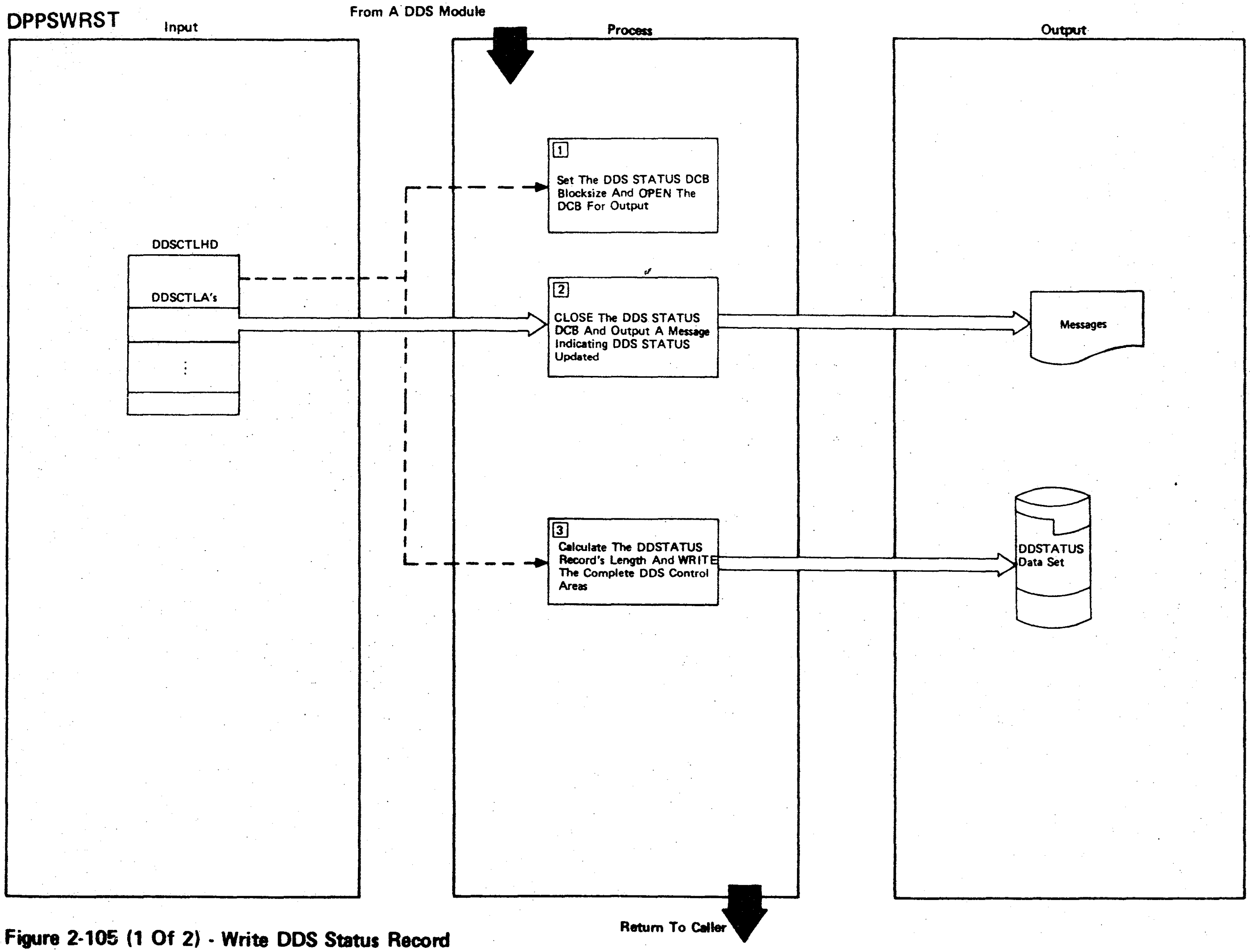
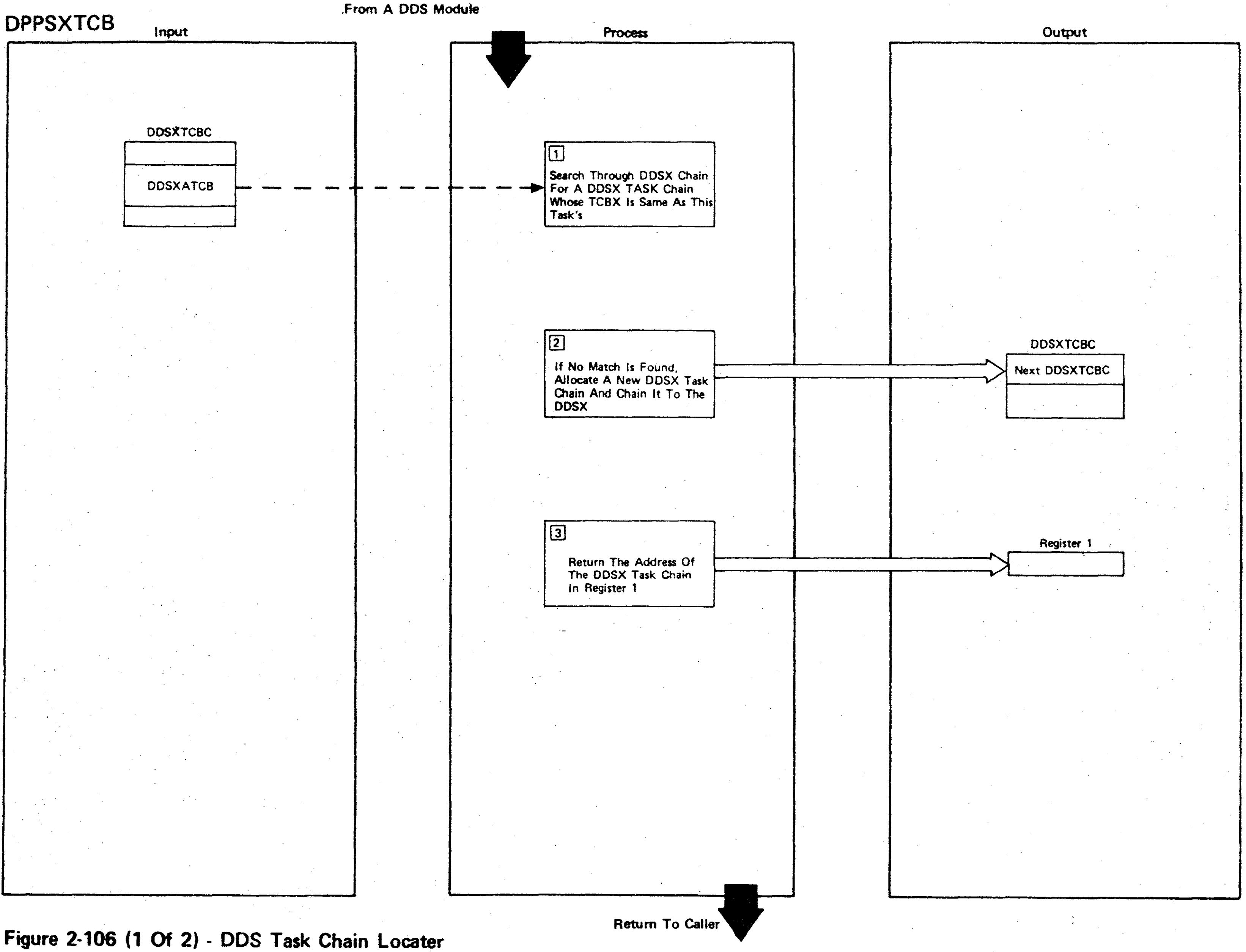

DPPSWRST
Input
From A DDS Module
Process
Output
1
Set The DDS STATUS DCB Blocksize And OPEN The DCB For Output
DDSCTLHD
DDSCTLA's
2
CLOSE The DDS STATUS DCB And Output A Message Indicating DDS STATUS Updated
Messages
3
Calculate The DDSTATUS Record's Length And WRITE The Complete DDS Control Areas
DDSTATUS Data Set
Return To Caller

Figure 2-105 (1 Of 2) - Write DDS Status Record

Figure 2-105 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If this is read only mode, output a message indicating that DDSTATUS WRITE cannot be executed, and if OPEN failed, output a message indicating such.	DPP884I DPP880I	DPPSWRST
2	Each DDS control area (DDSCTLA) should be output to contain all DDS declarations. If SYNAD occurs, output message indicating unable to update DDSTATUS.	DPP881I	DPPSWRST
3	The message DDSTATUS HAS BEEN UPDATED will notify the operator that the DDS declarations have changed.	DPP885I	DPPSWRST

DPPSXTCB



1 Search Through DDSX Chain For A DDSX TASK Chain Whose TCBX Is Same As This Task's

2 If No Match Is Found, Allocate A New DDSX Task Chain And Chain It To The DDSX

3 Return The Address Of The DDSX Task Chain In Register 1

Figure 2-106 (1 Of 2) - DDS Task Chain Locater

Figure 2-106 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The DDSXTCBCs are chained together and each one contains the address of its TCBX.		DPPSXTCB
2	Allocation of a DDSXTCBC is obtained only once per task, and it is initialized at allocation time.		DPPSXTCB
3	The address of the DDSXTCBC, either found or recently allocated, is returned to the user.		DPPSXTCB

Supplementary Services

The Supplementary Services functional area is composed of individual subroutines each of which is entered by a macro call. These routines are not logically associated with any of the other functional areas but are used as subroutines by most of the functional areas.

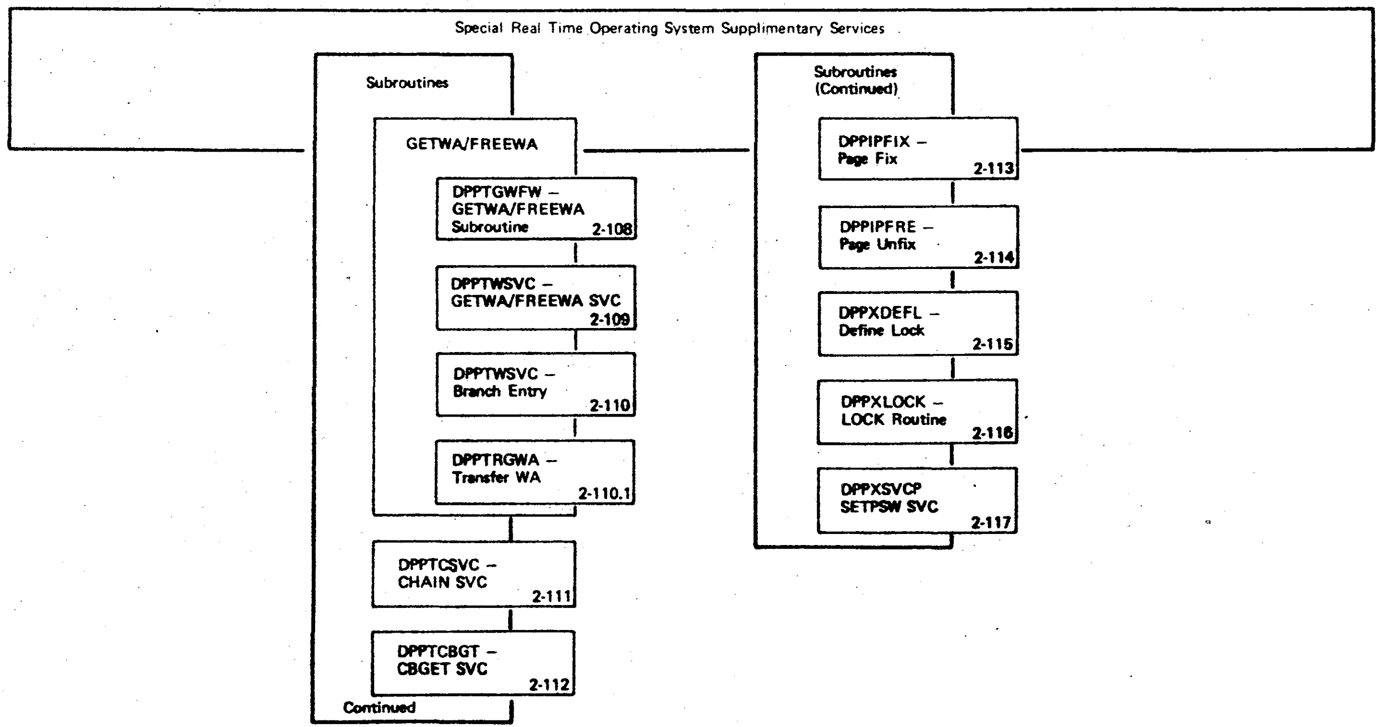

Special Real Time Operating System Supplimentary Services
Subroutines
GETWA/FREEWA
DPPTGWFW – GETWA/FREEWA Subroutine 2-108
DPPTWSVC – GETWA/FREEWA SVC 2-109
DPPTWSVC – Branch Entry 2-110
DPPTRGWA – Transfer WA 2-110.1
DPPTCSVC – CHAIN SVC 2-111
DPPTCBGT – CBGET SVC 2-112
Continued
Subroutines (Continued)
DPPIPFIX – Page Fix 2-113
DPPIPFRE – Page Unfix 2-114
DPPXDEFL – Define Lock 2-115
DPPXLOCK – LOCK Routine 2-116
DPPXSVCP SETPSW SVC 2-117

Figure 2-107 Special Real Time Operating System Supplementary Services Overview

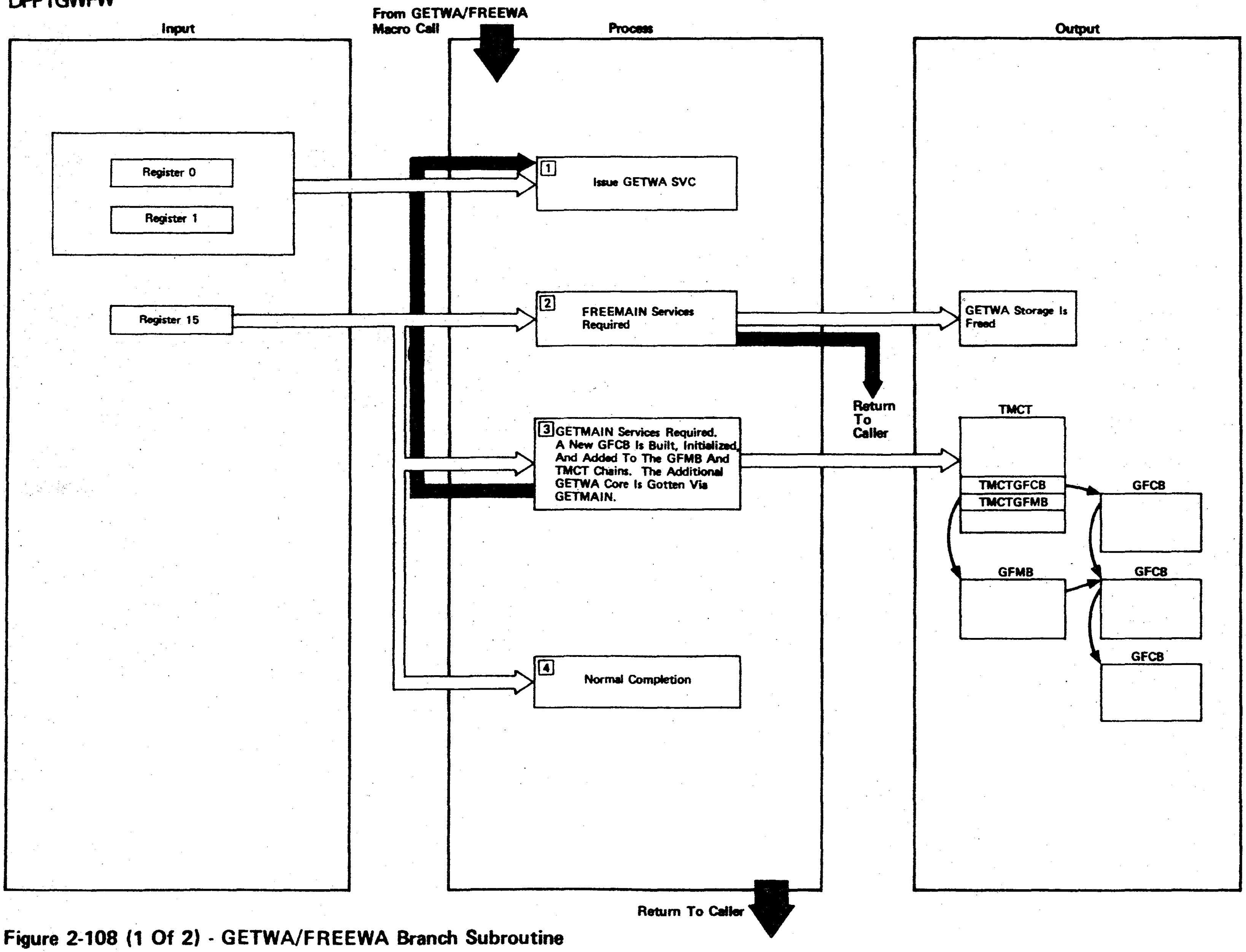

From GETWA/FREEWA Macro Call
Input
Process
Output
Register 0
Register 1
Register 15
1 Issue GETWA SVC
2 FREEMAIN Services Required
3 GETMAIN Services Required. A New GFCB Is Built, Initialized, And Added To The GFMB And TMCT Chains. The Additional GETWA Core Is Gotten Via GETMAIN.
4 Normal Completion
Return To Caller
GETWA Storage Is Freed
TMCT
TMCTGFCB
TMCTGFMB
GFMB
GFCB
GFCB
GFCB
Return To Caller

Figure 2-108 (1 Of 2) - GETWA/FREEWA Branch Subroutine

Figure 2-108 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	A GETWA SVC is issued.		DPPTGWFW
2	FREEMAIN services are indicated by register 15 being negative and equal to register 1. Register 1 contains the complement of the address to be freed, and register 0 contains the length. The FREEMAIN is issued and then control is returned to the user.		DPPTGWFW
3	GETMAIN services are indicated by register 15 being negative and not equal to register 1. The high-order byte of register 1 contains $\underbrace{0\ 1}_{\text{GETWA TYPE}}\ \underbrace{2\ 3\ \ 4\ 5\ 6\ 7}_{\text{ID of GFMB needing expanding}}$ GETWA TYPE - 00 - AP 01 - AT 10 - PC Additional GETWA space is obtained, a GFCB is created and initialized, and the GETWA is retried. If CBGET core could not be obtained, the GETWA storage is freed, and return code 8 is passed to the user.		DPPTGWFW
4	If register 15 is not negative, return is passed to the caller.		DPPTGWFW

DPPTWSVC

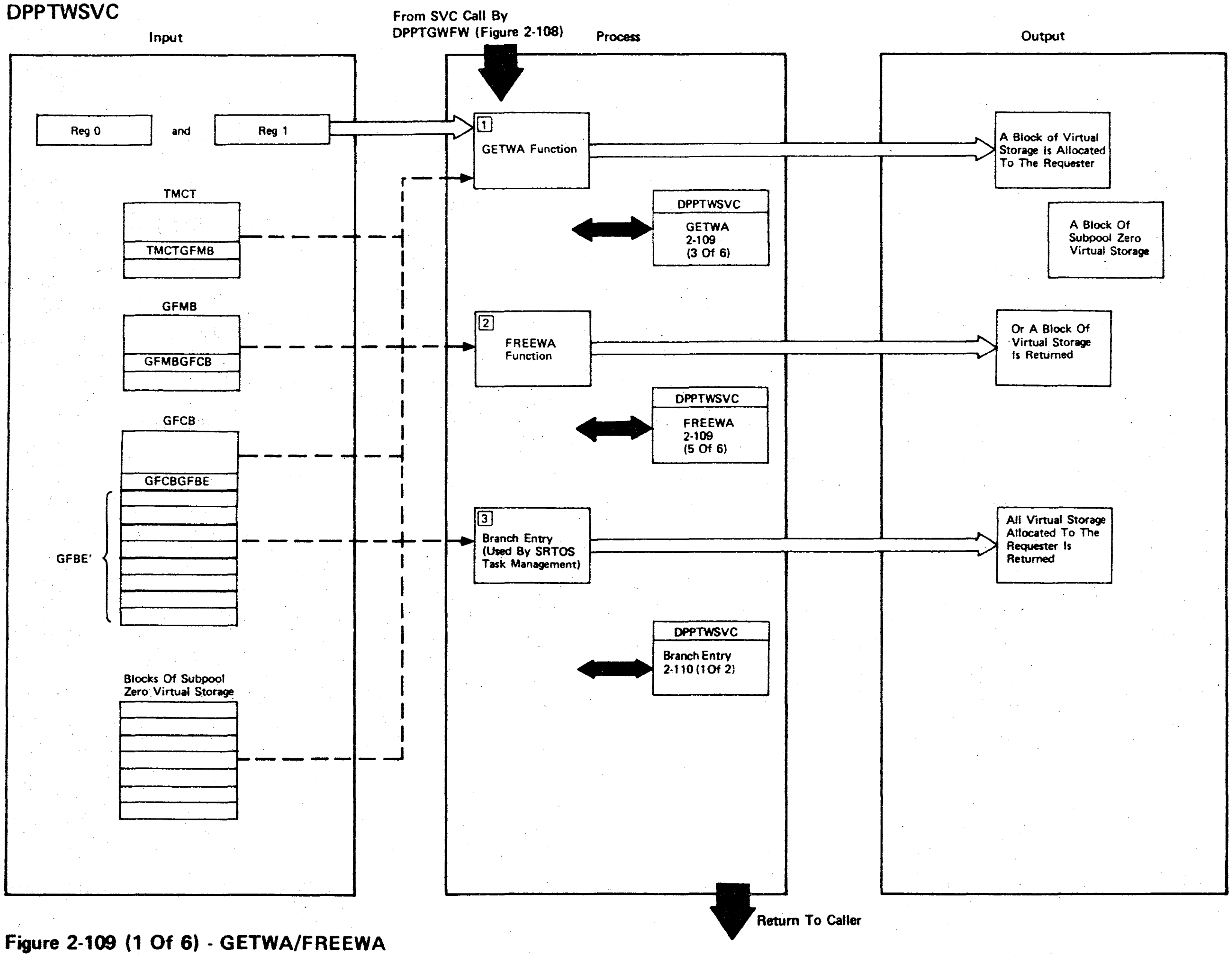


Figure 2-109 (1 Of 6) - GETWA/FREEWA

Figure 2-109 (2 of 6).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The function (GET or FREE) is determined by register 1. If negative, the request is a GETWA. If positive, it is a FREEWA. A detailed description of the GETWA function is shown in Figure 2-109 (3 of 6).		DPPTWSVC DPTWSVC1
2	A detailed description of FREEWA is shown in Figure 2-109 (5 of 6).		DPPTWSVC DPTWSVC1
3	The branch entry to GETWA is used by Special Real Time Operating System task management only (DPPTPMON). It is used to free all AP and AT type GETWA storage allocated to a task. A detailed description of the branch entry function is shown on Figure 2-110.		DPTWSVC3

DPPTWSVC

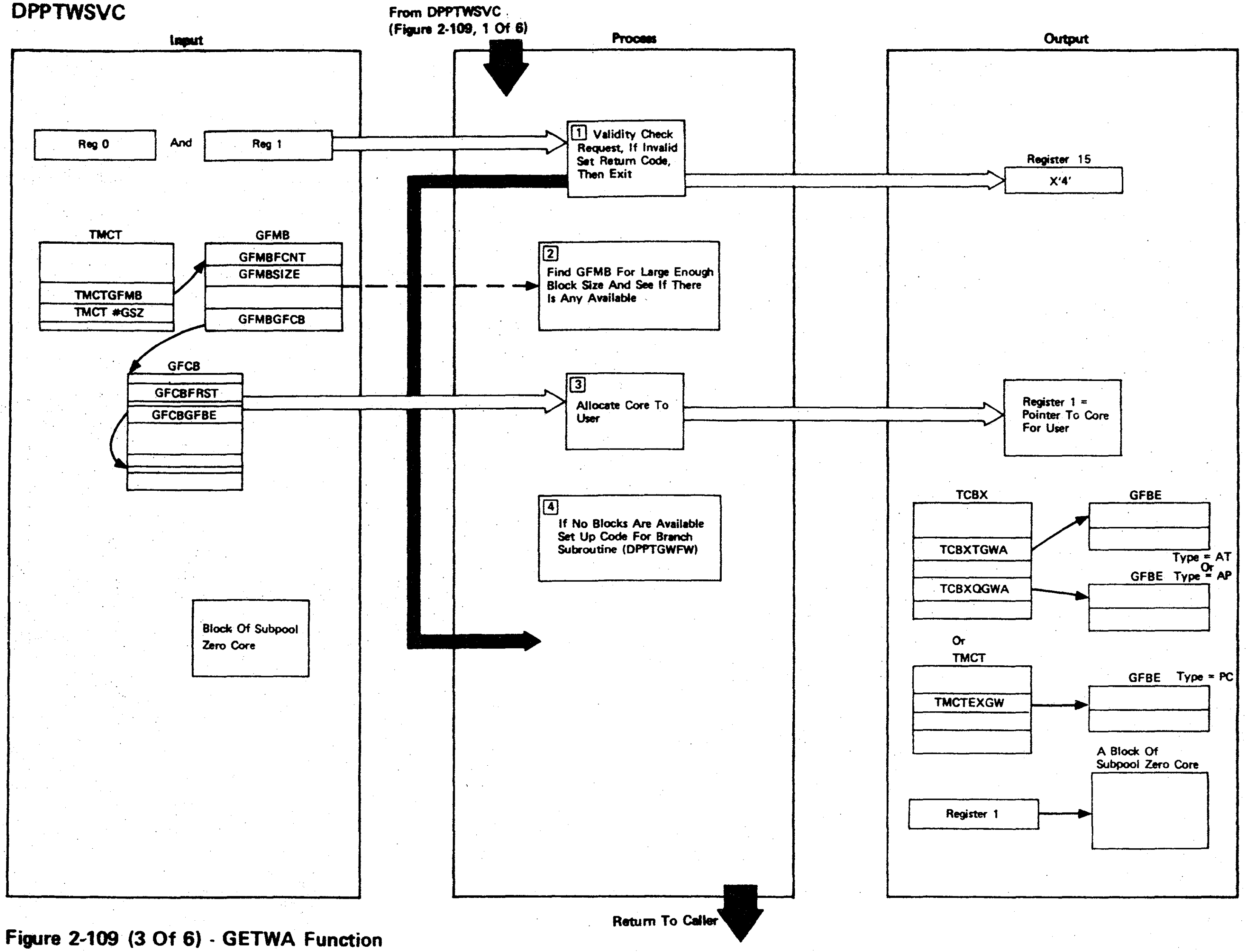

From DPPTWSVC
(Figure 2-109, 1 Of 6)
Input
Process
Output
Reg 0
And
Reg 1
1 Validity Check Request, If Invalid Set Return Code, Then Exit
Register 15
X'4'
TMCT
TMCTGFMB
TMCT #GSZ
GFMB
GFMBFCNT
GFMBSIZE
GFMBGFCB
2 Find GFMB For Large Enough Block Size And See If There Is Any Available
GFCB
GFCBFRST
GFCBGFBE
3 Allocate Core To User
Register 1 = Pointer To Core For User
4 If No Blocks Are Available Set Up Code For Branch Subroutine (DPPTGWFW)
Block Of Subpool Zero Core
TCBX
TCBXTGWA
TCBXQGWA
GFBE
Type = AT
Or
GFBE Type = AP
Or
TMCT
TMCTEXGW
GFBE Type = PC
A Block Of Subpool Zero Core
Register 1
Return To Caller

Figure 2-109 (3 Of 6) - GETWA Function

Figure 2-109 (4 of 6).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The size of the request is validity checked to ensure that the request is not for zero bytes. A zero request will not be honored, and a return code of 4 will be returned.		DPPTVSVC
2	Each GFMB is checked to see the block size it represents. If the blocksize (GFMBSIZE) is equal to or greater than the requested size, the number of blocks available (GFMBFCNT) is checked. If there is a block available, a GFCB for the size with available blocks is located (GFMBGFCB) and a GFBE is taken from the free chain (GFCBGFBE). The core address for the core represented by this GFBE is calculated.		DPTWSVC1
3	The GFBE for the core being allocated is then chained to the user TCBXTCWA for Type = AT TCBXQGWA for Type = AP TMCTEXGW for Type = PC The core address is placed in register 1. If a positive return code is being passed to the requestor, register 1 will be set to negative 1.		DPTWSVC1
4	If no core is available, register 15 is made negative and the high-order byte of register 1 is set up with a code for the type and size as follows: High-order byte bits $0\ \underbrace{1\ 2}\quad\underbrace{3\ 4\ 5\ 6\ 7}$ 0 = AP GFMB ID for size 1 = AT 2 = PC		

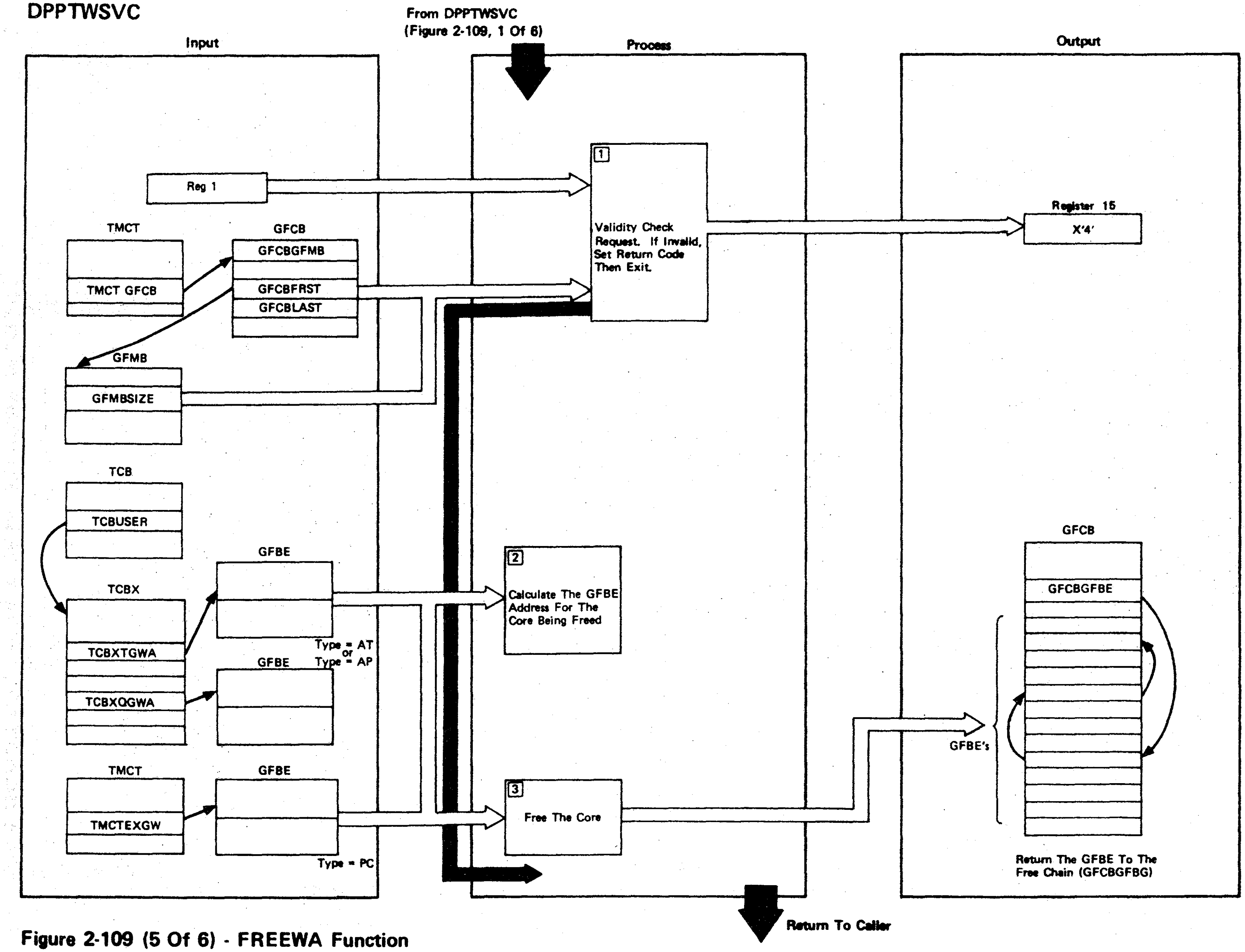


Figure 2-109 (5 Of 6) - FREEWA Function

Figure 2-109 (6 of 6).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The GFCB that represents the storage to be freed (address falls within GFCBFRST-GFCBLAST) is located. If the address is not represented by a GFCB, it is invalid. The GFMB is then located, and the address is checked to ensure it falls on a block boundary for the blocksize (address/GFMBSIZE with no remainer). If it is not on a block boundary, it is invalid.		DPPTWSVC
2	The GFBE is located for the specified core.		DPPTWSVC
3	If the allocated bit is on in the GFBE, the GFBE is dechained from its existing chain and added in to the GFCB free chain (GFCBGFBE). If the allocated bit is not on, it is an invalid free request and the requestor is given a return code 4. If all blocks in the GFCB are now free, the GFCB is moved to the end of the GFMB chain. If the GFCB is not the initial allocation and all blocks are free and the total free count is larger than initial allocation and free count for this GFCB, the GFCB is dechained and freed. The compliment of the GETWA address is placed in registers 1 and 15 so that the branch subroutine (DPPTGWFW) will FREEMAIN the GETWA storage.		DPPTWSVC

DPPTWSVC

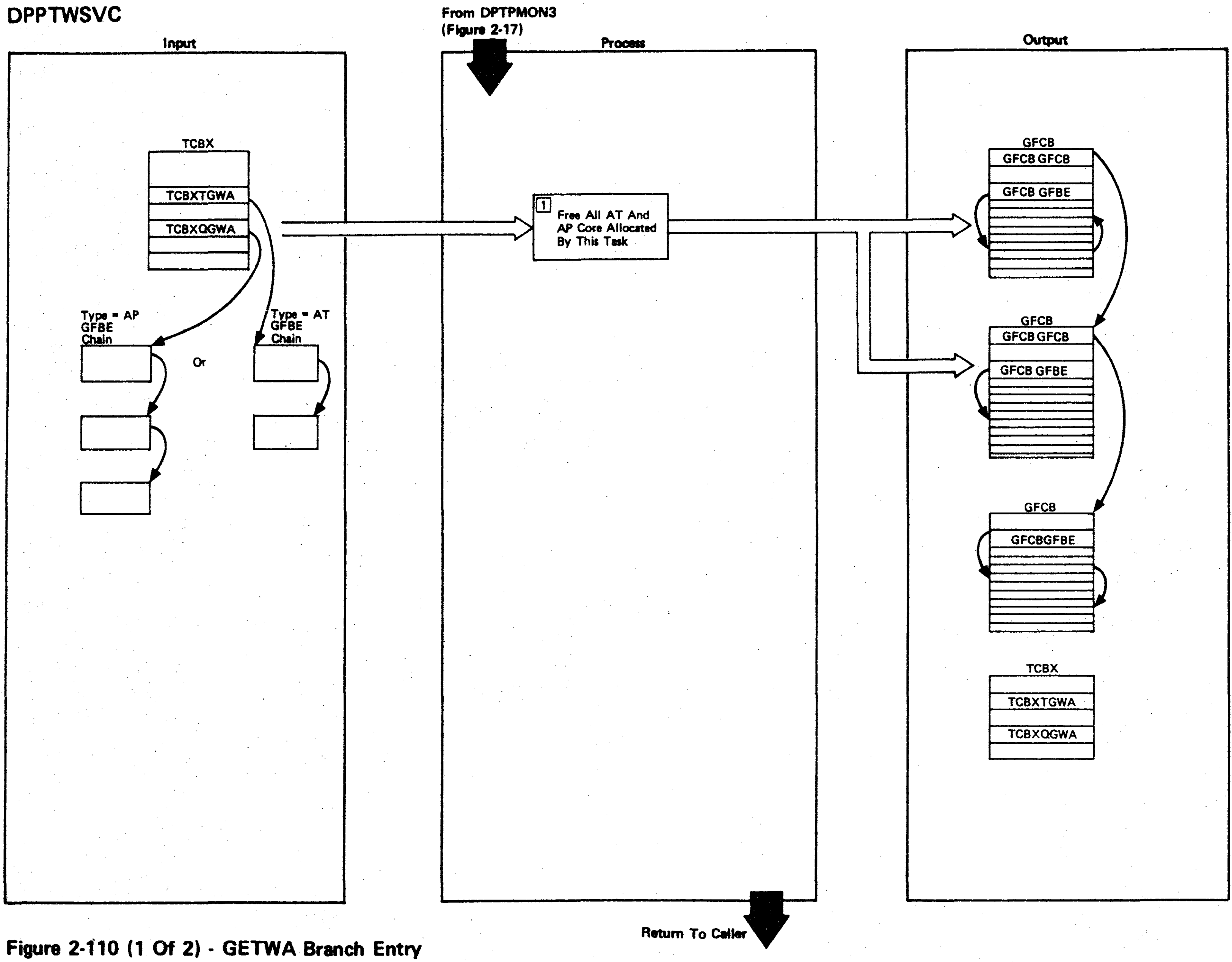


Figure 2-110 (1 Of 2) - GETWA Branch Entry

Figure 2-110 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The GFBE is dechained from its existing chain. The GFBE ID field is used to calculate the GFMB which represents the core size. The GFCB address is obtained (GFMBGFCB), and the GFCB owning the block is located and the GFBE returned to the GFCB free chain (GFCBGFBE). The address of the chain to be processed is passed to the branch entry by DPPTPMON in register 1. The processing terminates when this address points to itself (all GFBEs are removed from the chain). If all blocks in the GFCB are now free, the GFCB is mould to the end of the GFMB chain. If the GFCB is not the initial allocation and all blocks are free and the total free count is larger than initial allocation and free count for this GFCB, the GFCB is dechained and freed. The storage represented by this GFCB is then FREEMAINed.		DPTWSVC3

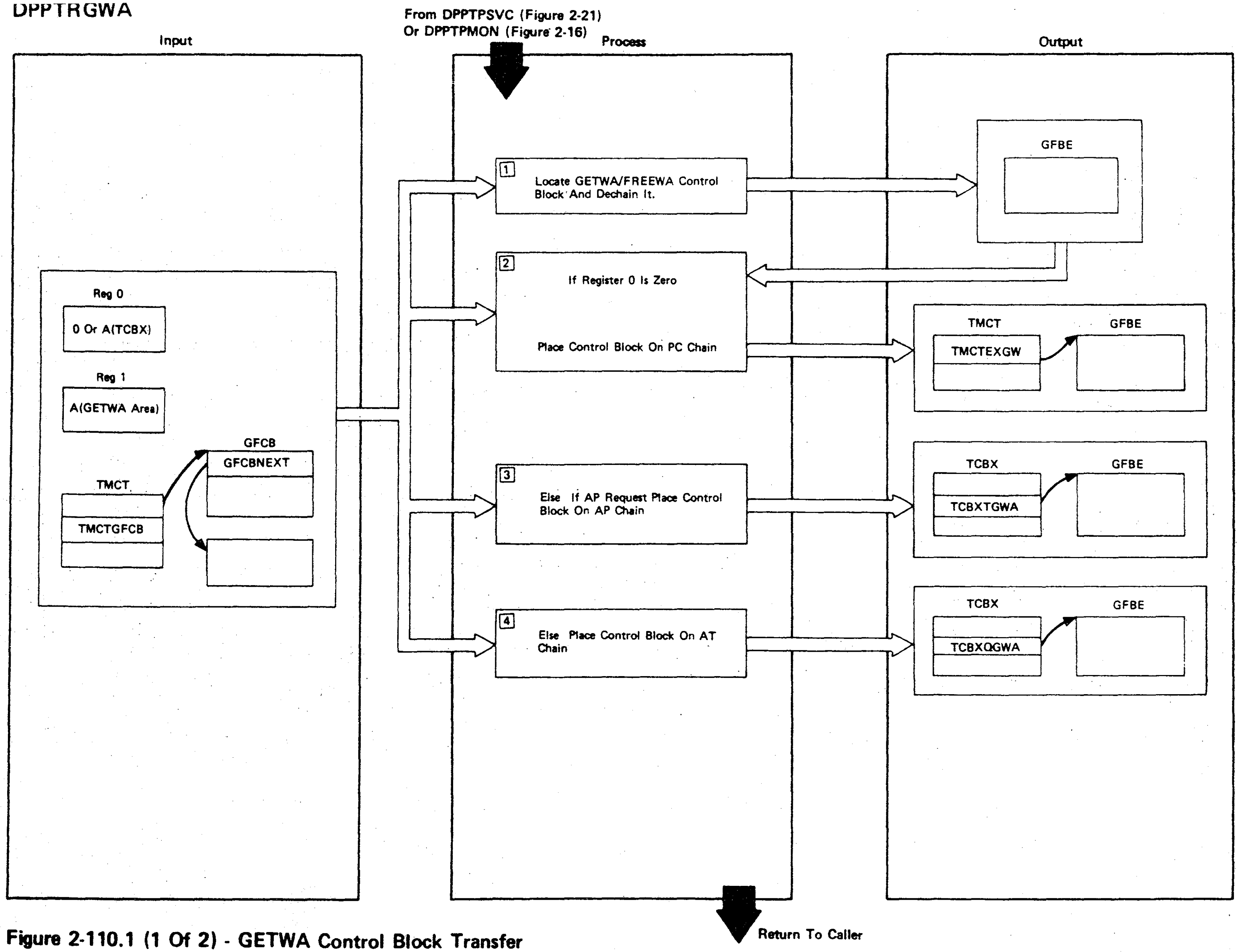

From DPPTPSVC (Figure 2-21)
Or DPPTPMON (Figure 2-16)
Input
Process
Output
Reg 0
0 Or A(TCBX)
Reg 1
A(GETWA Area)
GFCB
GFCBNEXT
TMCT
TMCTGFCB
1
Locate GETWA/FREEWA Control Block And Dechain It.
2
If Register 0 Is Zero
Place Control Block On PC Chain
3
Else If AP Request Place Control Block On AP Chain
4
Else Place Control Block On AT Chain
GFBE
TMCT
TMCTEXGW
GFBE
TCBX
TCBXTGWA
GFBE
TCBX
TCBXQGWA
GFBE
Return To Caller

Figure 2-110.1 (1 Of 2) - GETWA Control Block Transfer

Figure 2-110.1 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The GFCBs are scanned to locate which one controls the GETWA area passed in register 1.		DPPTRGWA
2	A zero in register 0 indicates that this area is to be placed on the PC chain.		DPPTRGWA
3	A 4 in the high-order byte of register 1 indicates that the area is to be placed on the AP chain.		DPPTRGWA
4	An 8 in the high-order byte of register 1 indicates that this area is to be placed on the AT chain.		DPPTRGWA

DPPTCSVC

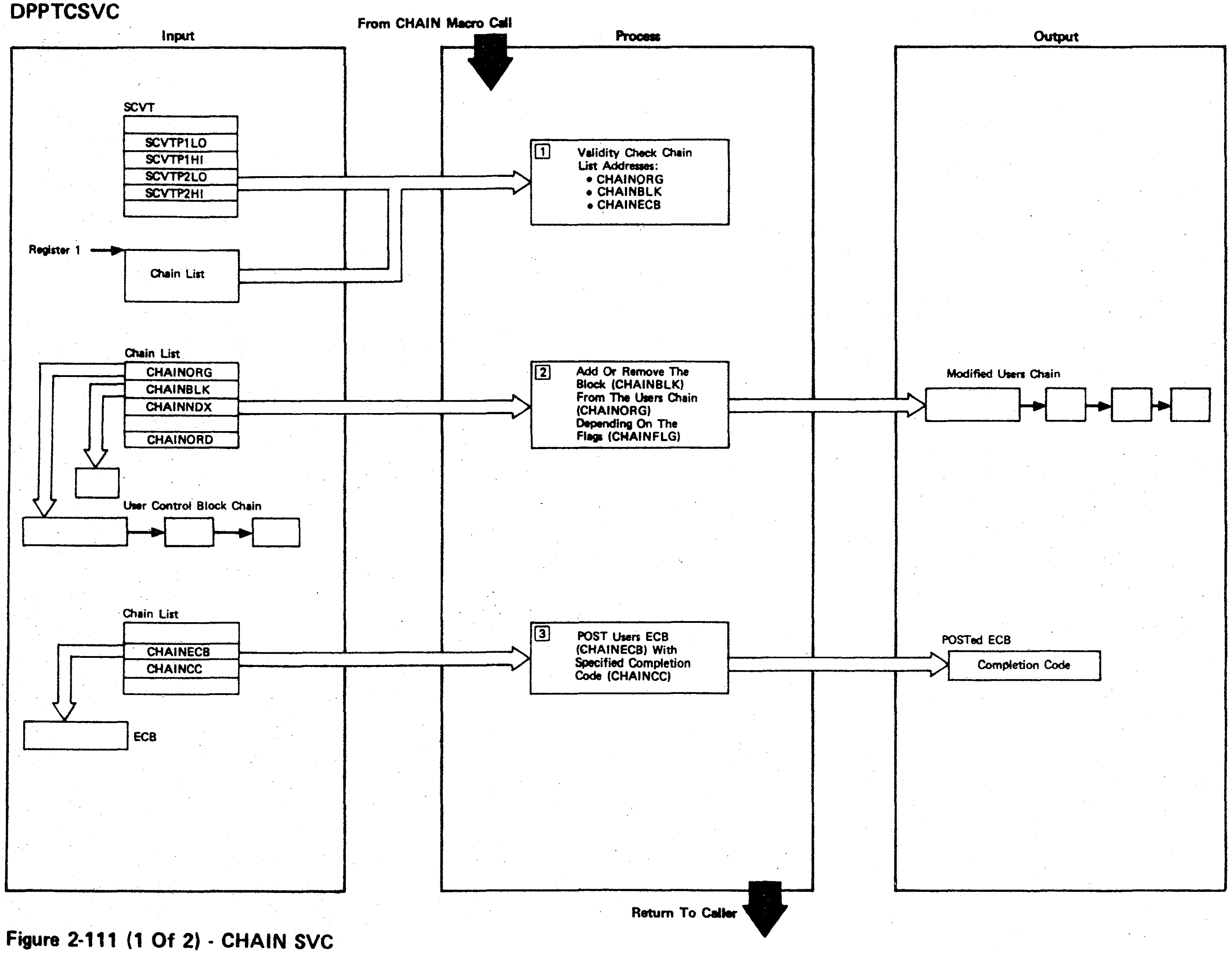


Figure 2-111 (1 Of 2) - CHAIN SVC

Figure 2-111 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The addresses passed to **CHAIN** in the chain list (CHAINORG, CHAINBLK, CHAINECB) are checked. They must be within the partition (SCVTP1HI, SCVTP1LO). If two partition operation, they must be within either partition (SCVTP2HI, SCVTP2LO).		DPPTCSVC
2	The block passed (CHAINBLK) is added to or removed from the specified chain (CHAINORG). If the first and add flags are on, the new block is added at the origin. If the add flag is on, the chain is scanned, and the new block is added to the end. If the order flag is on, the new block is inserted in the chain in ascending order by the value in the CHAINORD field. The chaining pointers in the blocks are displaced in the blocks by the value in CHAINNDX. If the add flag is off, the block is removed from the chain.		DPTCSVC1
3	The specified ECB is posted with the specified completion code from the chain list (CHAINECB) if the operation completed successfully, otherwise, the user is given a nonzero return code.		

DPPTCBGT

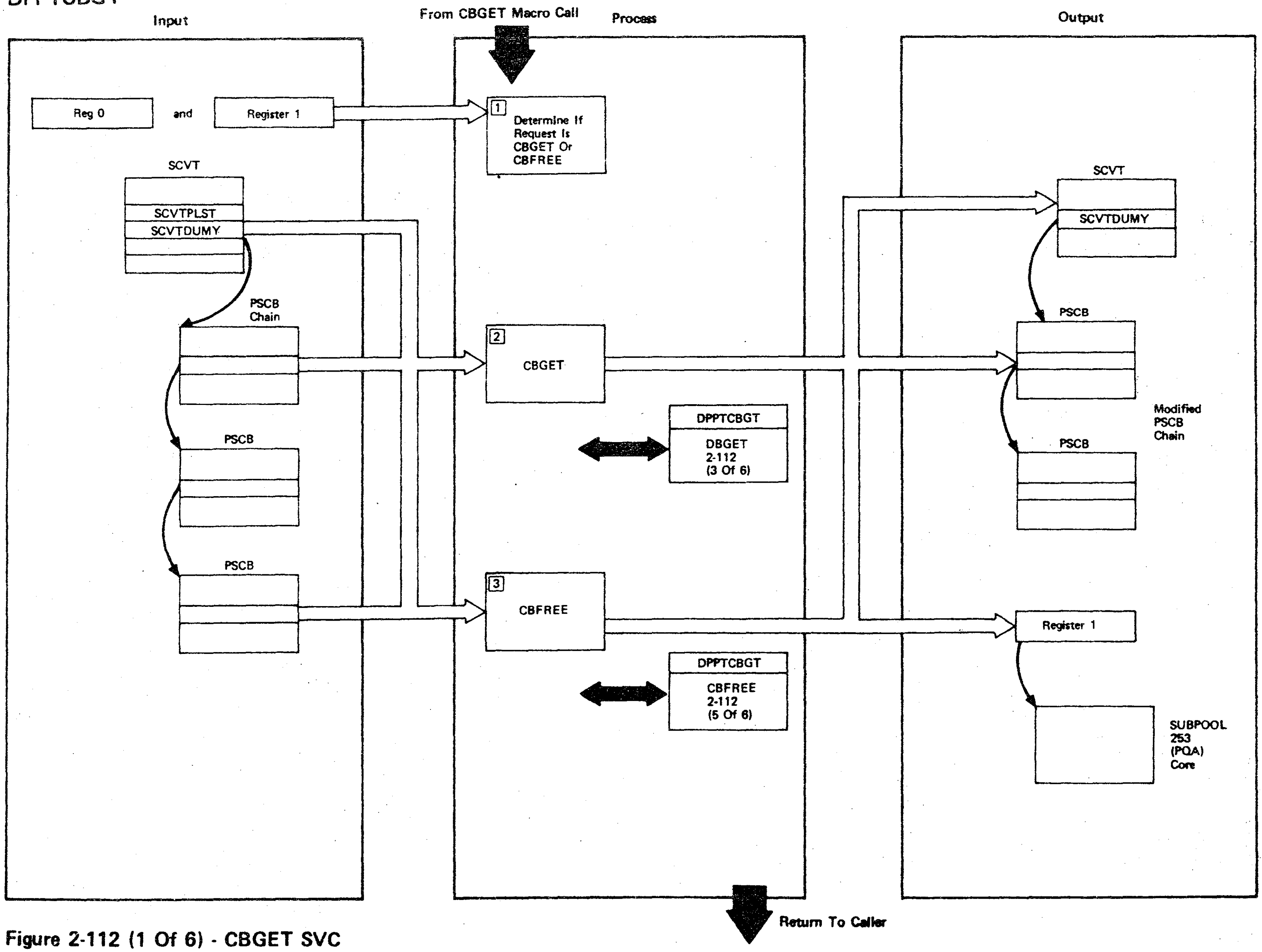


Figure 2-112 (1 Of 6) - CBGET SVC

Figure 2-112 (2 of 6).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Register 1 is checked to determine if the request is a CBGET or CBFREE. If register 1 is negative, the request is CBGET; if positive, the request is CBFREE.		DPPTCBGT
2	Blocks of protected storage (PQA Subpool 253) are allocated to the requestors in 32-byte multiples (i.e., a request for 50 bytes will allocate 64). The requestor can have any PSW protect key; however, if he does not have zero, he will get a protection interrupt when he attempts to store in the allocated storage. The core will be cleared to zeros before it is allocated to the requestor.		DPPTCBGT
3	Blocks are returned to the pool of PQA by the CBFREE function. The freed core will be combined with any adjacent free core at the time it is freed.		

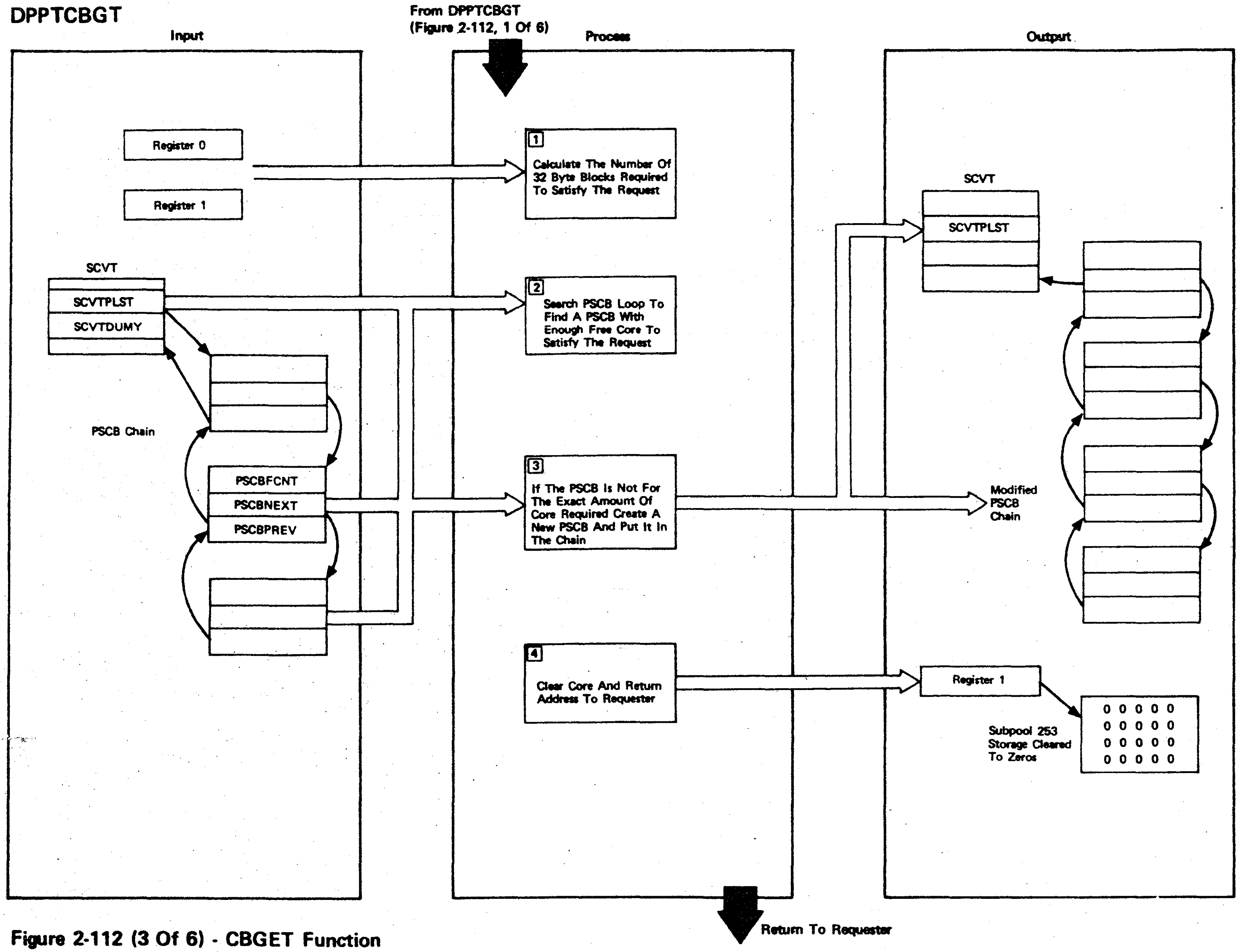

DPPTCBGT
From DPPTCBGT
(Figure 2-112, 1 Of 6)
Input
Process
Output
Register 0
Register 1
1
Calculate The Number Of 32 Byte Blocks Required To Satisfy The Request
SCVT
SCVTPLST
SCVTDUMY
2
Search PSCB Loop To Find A PSCB With Enough Free Core To Satisfy The Request
PSCB Chain
PSCBFCNT
PSCBNEXT
PSCBPREV
3
If The PSCB Is Not For The Exact Amount Of Core Required Create A New PSCB And Put It In The Chain
SCVT
SCVTPLST
Modified PSCB Chain
4
Clear Core And Return Address To Requester
Register 1
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
Subpool 253 Storage Cleared To Zeros
Return To Requester

Figure 2-112 (3 Of 6) - CBGET Function

Figure 2-112 (4 of 6).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The number of 32 byte blocks is determined by adding the size of a PSCB plus a rounding factor to the request size and dividing by 32- $\frac{\text{(request size + PSCBLNTH + 31)}}{32}$.		DPPTCBGT
2	The PSCB loop is entered at the block pointed to by the SCVT last used pointer (SCVTPLST). A stop indicator bit is turned on so that the program will know when it has scanned the entire loop. The loop is scanned until a PSCB representing a number of 32 byte blocks equal to or greater than the requested number is found.		DPPTCBGT
3	If the PSCB has the exact number of blocks required to satisfy the request, the storage is allocated by complementing PSCB free count field (PSCBFCNT), clearing the requested length of core, and returning the starting address to the requestor. If the PSCB contains more free blocks than are required, a new PSCB is built just above the core required to satisfy the request, and the new PSCB is inserted into the circular PSCB chain. The new PSCB will represent a number of free blocks equal to the number in the old PSCB number required to satisfy the request. If the stop indicator is reached, there is not enough core to satisfy the request. The requestor is passed a return core of 4. The stop indicator is turned off and the last used pointer is updated.		DPPTCBGT
4	The core which has been allocated to a requestor is cleared to binary zeros for a length equal to the requested length the address is returned to the requestor in register 1.		DPPTCBGT

DPPTCBGT

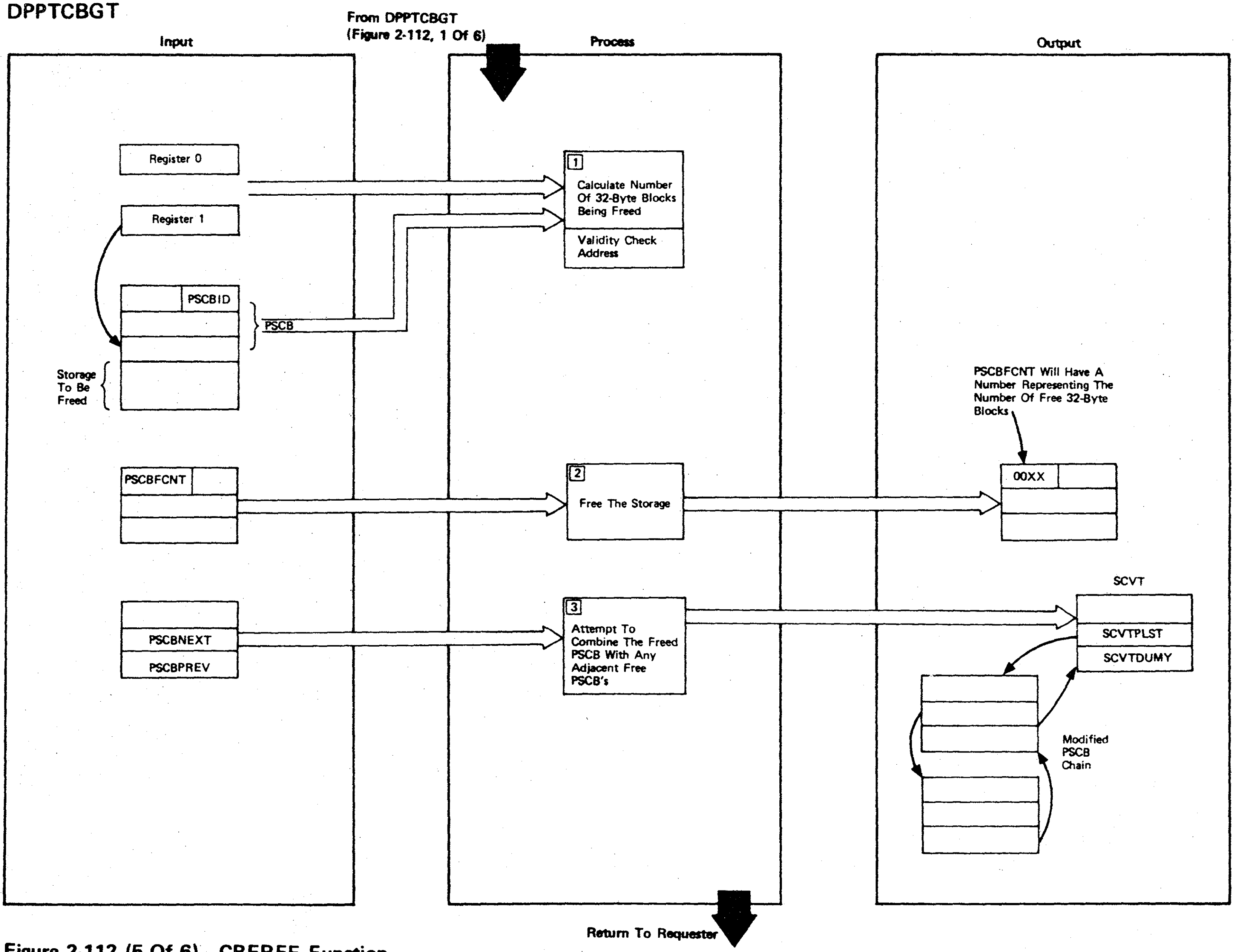


Figure 2-112 (5 Of 6) - CBFREE Function

Figure 2-112 (6 of 6).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The address passed to be freed (in register 1) is backed up by the PSCBLNTH value (a PSCB is built immediately preceding the core it represents). A check is made for the PSCBID field (hexadecimal 'C9C4'). If the ID is valid, the address to be freed is valid.		DPPTCBGT
2	The storage being freed by this request is freed by storing the number of blocks being freed in the PSCBFCNT field.		DPPTCBGT
3	The next PSCB in the circular chain (PSCBNEXT) is interrogated to determine if it represents free core (PSCBFCNT nonzero). If it does, the number of blocks it represents is added to the current PSCB and the PSCBNEXT PSCB is removed from the circular chain. If it represents allocated core (PSCBFCNT = 0), the chain is not modified. The previous PSCB (PSCBPREV) in the circular chain is then checked to see if it represents free core. If it does, the number of free blocks in the current PSCB is added to the number of free blocks in the previous, and the current PSCB is removed from the circular chain. If the previous PSCB represents allocated core, the chain is unmodified.		DPPTCBGT

DPPIPFIX

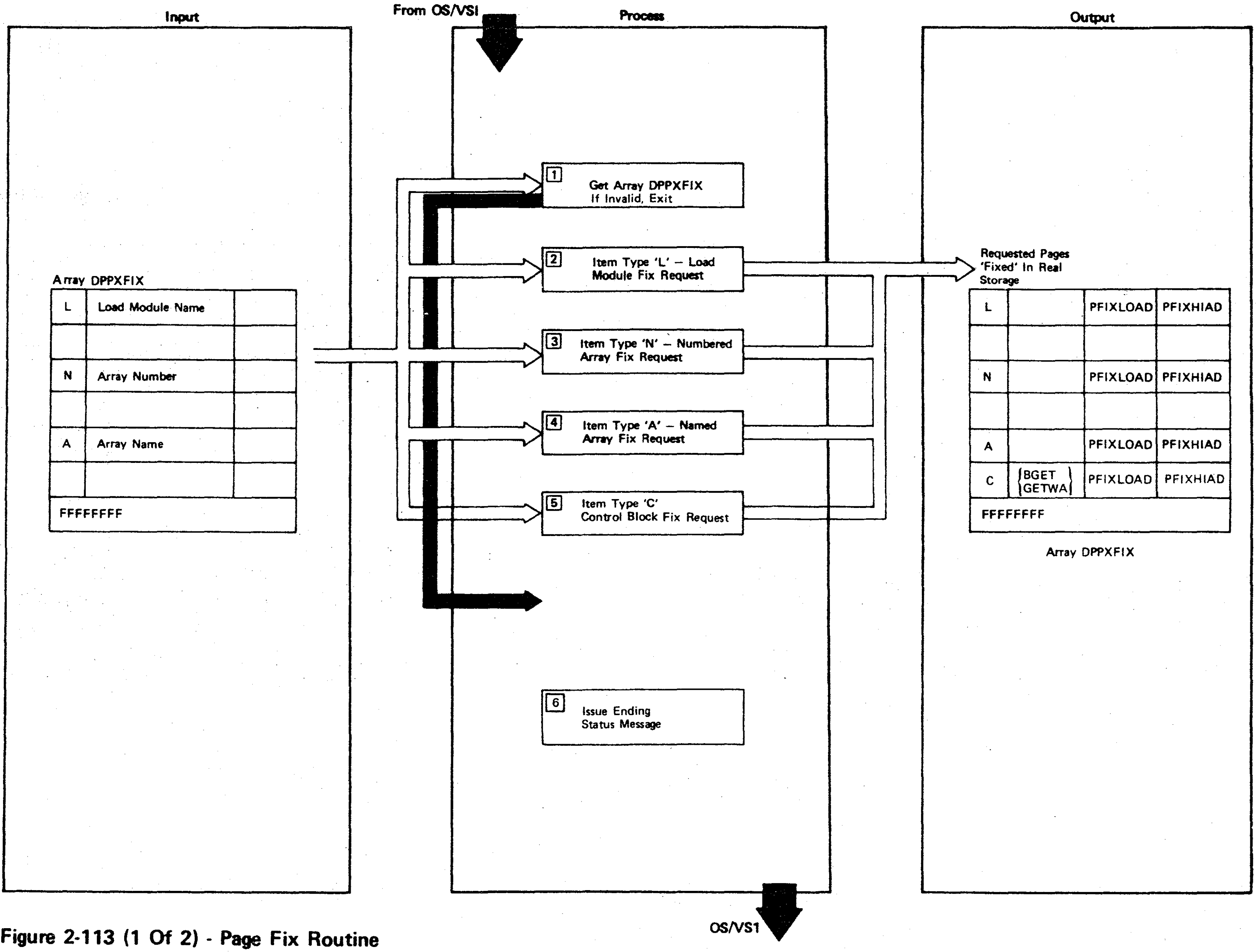
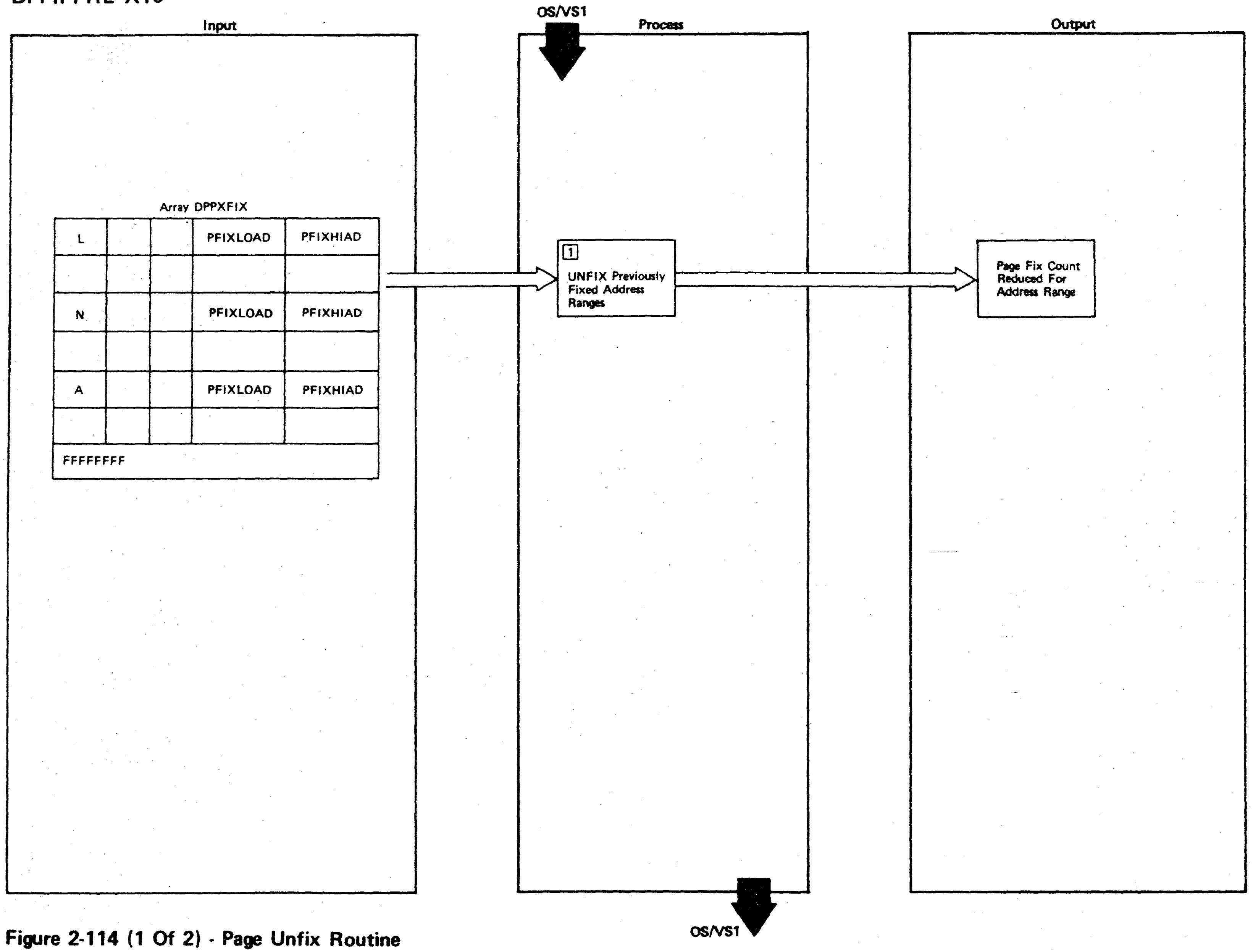


Figure 2-113 (1 Of 2) - Page Fix Routine

Figure 2-113 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Array DPPXFIX (VS array) is located by the GETARRAY macro. If the array cannot be found, the FIX routine terminates.	DPP047I	DPPIPFIX
2	A BLDL is issued for the load module; if it is found, the module is loaded. If a fix length is specified, the length is added to the entry point address to get the fix length. If no length was requested, the module length as returned by the LOAD is added to the entry point. In either case, the address range is fixed via the DPPFIX routine. If the FIX is successful, the addresses are stored in the DPPXFIX array (PFIXLOAD and PFIXHIAD).	DPP042I DPP043I	DPPIPFIX
3	The numbered array is located via GETARRAY. If a length was requested, it is added to the start address to get the range to be fixed; if not, the entire array is fixed. If the FIX is successful, the fixed address range is put in the array DPPXFIX.	DPP043I DPP048I	DPPIPFIX DPPIPFIX
4	Same as 3, for 'named' arrays.	DPP043I DPP048I	DPPIPFIX
5	The control block to be fixed is specified in the name field where 'CBGET' is a request for the CBGET storage & 'GETWA' is a request for the GETWA storage. If a length was requested it is added to the start address to get the range to be fixed, if not the entire control block is fixed. If the FIX is successful the FIXed address range is put in the array DPPXFIX.	DPP043I	DPPIPFIX
6	A message is issued stating whether or not all arrays and load modules were fixed successfully. A user ABEND will result if an invalid address range is specified.	DPP049I DPP052I USER 32	DPPIPFIX

DPPIPFRE X19


Input
OS/VS1
Process
Output
Array DPPXFIX
1
UNFIX Previously Fixed Address Ranges
Page Fix Count Reduced For Address Range
OS/VS1

Array DPPXFIX

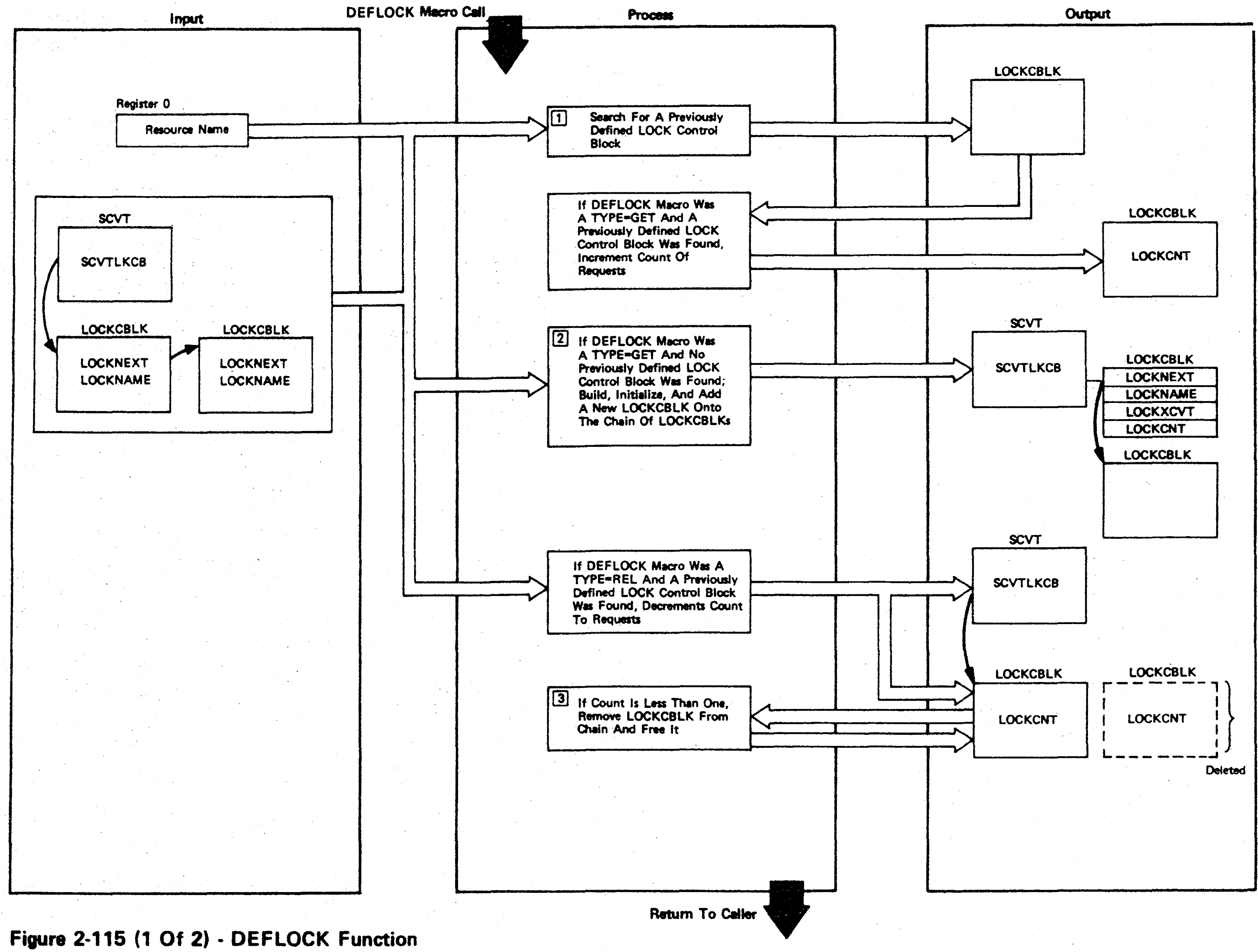
L			PFIXLOAD	PFIXHIAD
N			PFIXLOAD	PFIXHIAD
A			PFIXLOAD	PFIXHIAD
FFFFFFFF				

Figure 2-114 (1 Of 2) - Page Unfix Routine

Figure 2-114 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Array DPPXFIX is located with GETARRAY. A loop is set up to process each item in the array. If the PFIXHIAD is nonzero, the address range (PFIXLOAD - PFIXHIAD) is used with routine DPPFREE to unfix all previously fixed virtual storage.	DPP047I	DPPIPFRE

DPPXDEFL



1 Search For A Previously Defined LOCK Control Block

If DEFLOCK Macro Was A TYPE=GET And A Previously Defined LOCK Control Block Was Found, Increment Count Of Requests

2 If DEFLOCK Macro Was A TYPE=GET And No Previously Defined LOCK Control Block Was Found; Build, Initialize, And Add A New LOCKCBLK Onto The Chain Of LOCKCBLKs

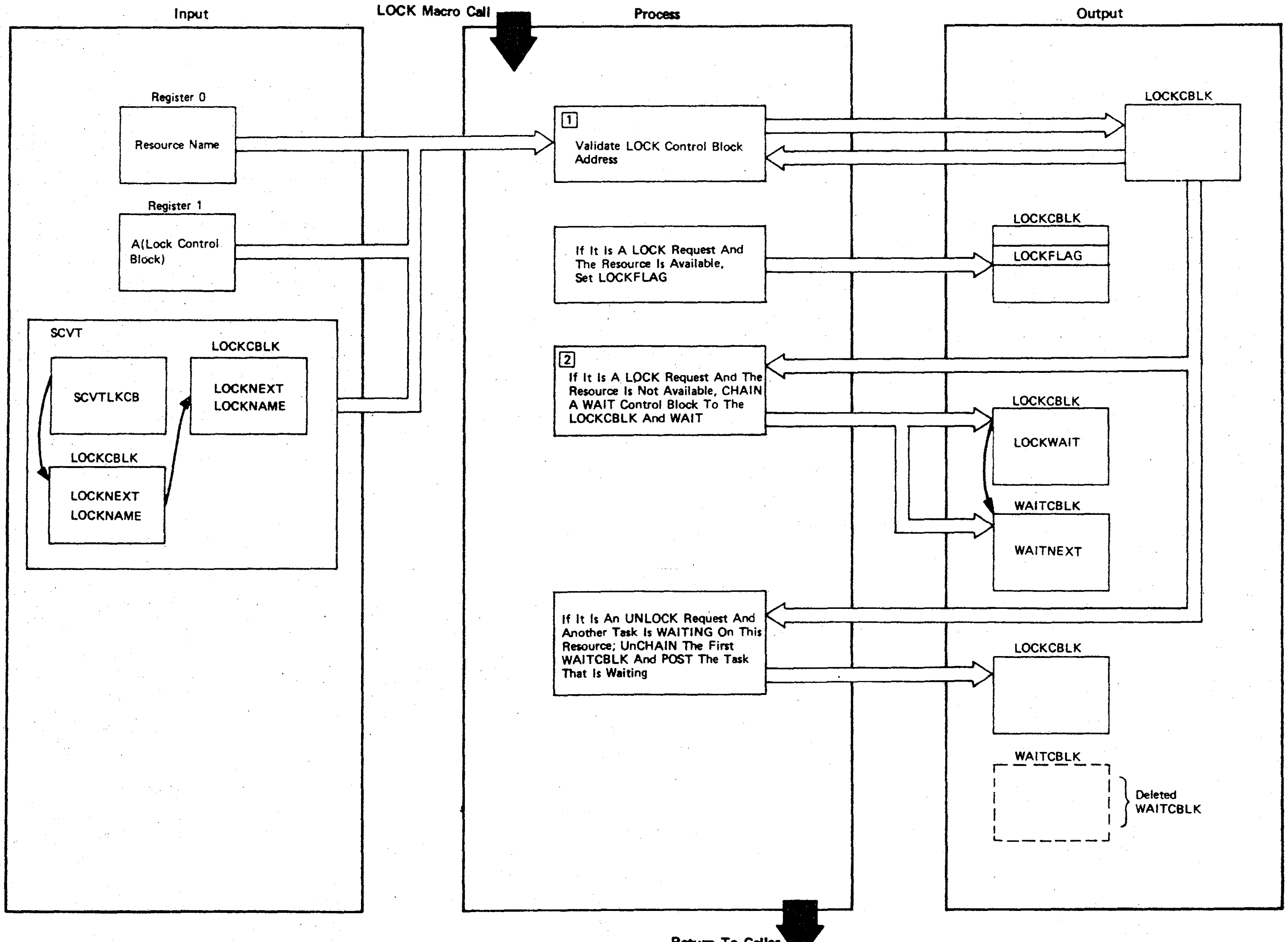
If DEFLOCK Macro Was A TYPE=REL And A Previously Defined LOCK Control Block Was Found, Decrements Count To Requests

3 If Count Is Less Than One, Remove LOCKCBLK From Chain And Free It

Figure 2-115 (1 Of 2) - DEFLOCK Function

Figure 2-115 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	On entry to DPPXDEFL, register 0 contains the resource name.		DPPXDEFL
2	A CHAIN macro call is used to add the LOCKCBLK to the chain of LOCKCBLKs.		DPPXDEFL
3	A CHAIN macro call is used to remove the LOCKCBLK from the chain of LOCKCBLKs.		DPPXDEFL



1 Validate LOCK Control Block Address

If It Is A LOCK Request And The Resource Is Available, Set LOCKFLAG

2 If It Is A LOCK Request And The Resource Is Not Available, CHAIN A WAIT Control Block To The LOCKCBLK And WAIT

If It Is An UNLOCK Request And Another Task Is WAITING On This Resource; UnCHAIN The First WAITCBLK And POST The Task That Is Waiting

Figure 2-116 (1 Of 2) - LOCK Function

Figure 2-116 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The resource name in register 0 must match the resource name in the LOCK control block pointed to by the address in register 1.		DPPXLOCK
2	The CHAIN macro is used to add the WAIT control block to the bottom of the WAITCBLK chain.		DPPXLOCK

DPPXSVCP

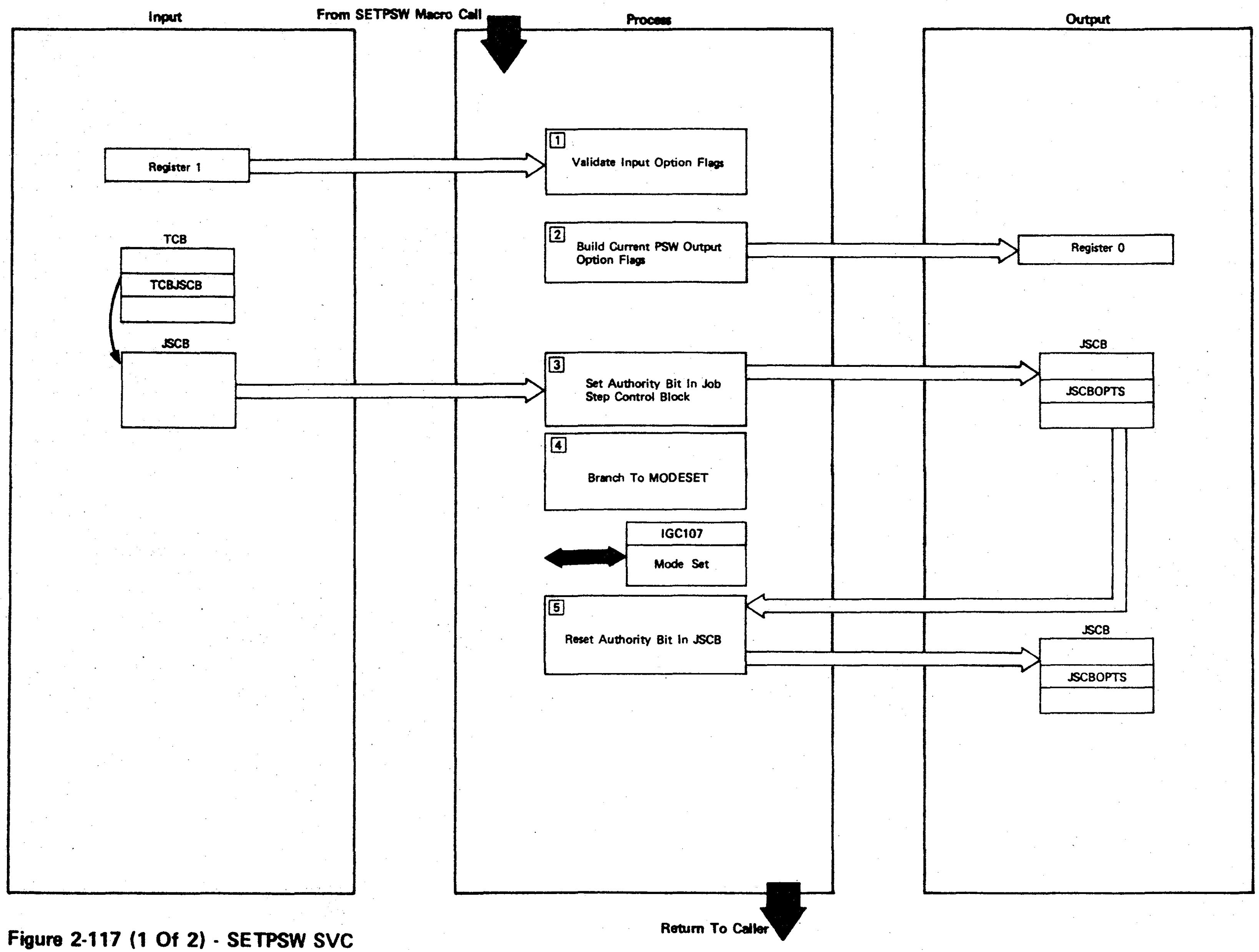


Figure 2-117 (1 Of 2) - SETPSW SVC

Figure 2-117 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The input option flag in register 1 is used to indicate the desired PSW setting. This flag must conform to the bit settings that the MODESET PSW recognizes. (See OS/VS1 Supervisor Logic SY24-5155.)		DPPXSVCP
2	DPPXSVCP builds an option flag based on the PSW setting at entry to DPPXSVCP. The user can then reset the PWS back to the original PSW by using these option flags as input option flags on another SETPSW SVC.		DPPXSVCP
3	Bit 7 of the JSCBOPTS field in the Job Step Control Block is set to allow this task to branch to the MODESET routine.		
4	MODESET is used to change the old PSW setting.		
5	Bit 7 of the JSCBOPTS field in the JSCB is reset to the state it was in at entry to DPPXSVC.		

High-Level Language Interfaces

The Special Real Time Operating System routines provide an interface to allow PL/I and FORTRAN users to use most of the services provided by the Special Real Time Operating System. The interface routines are independent of the compiler level or the optimizing compilers. Figure 2-118 lists the Special Real Time Operating System macros supported by the interface routines for PL/I. The macros in the following table are also supported for FORTRAN, but there are no default structures.

Macro Name	ID	PL/I	
		Structure Name	Member Name
PATCH	0	PATCHSTR	PATCHDEF
PATCH Param	0	PARMSTR	PARMDEF
PTIME	4	PTIMESTR	PTIMEDEF
PTIME	4	PTIMRSTR	PTIMRDEF
DPATCH	8	DPACHSTR	DPACHDEF
REPATCH	12	REPCHSTR	REPCHDEF
GETARRAY	16	ARRAYSTR	ARRAYDEF
GETITEM	20	ITEMSTR	ITEMDEF
GETBLOCK	24	BLOCKSTR	BLOCKDEF
PUTARRAY	16	ARRAYSTR	ARRAYDEF
PUTITEM	20	ITEMSTR	ITEMDEF
PUTBLOCK	24	BLOCKSTR	BLOCKDEF
MESSAGE	40	MESAGSTR	MESAGDEF
PUTLOG	44	PTLOGSTR	PTLOGDEF
GETLOG	48	GTLOGSTR	GTLOGDEF
DUMPLOG	52	DPLOGSTR	DPLOGDEF
RECORD	56	RECRDSTR	RECRDDEF
PATCH WAIT	60	WAITSTR	WAITDEF

Figure 2-118. Macro Supported by FORTRAN-PL/I Interface Routines

All interface routines are invoked as shown in Figure 2-119. The parameters are passed using standard linkage conventions to the assembler language interface routine. The interface routine adjusts the parameter list and then issues an execute form of the appropriate macro to invoke the desired service. After the service routine has completed execution, the interface routine stores the return code for use by the calling program and returns to the caller.

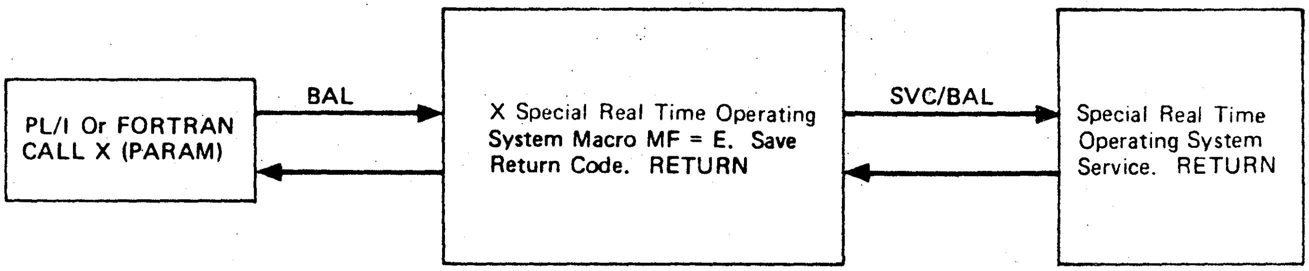

PL/I Or FORTRAN
CALL X (PARAM)
BAL
X Special Real Time Operating System Macro MF = E. Save Return Code. RETURN
SVC/BAL
Special Real Time Operating System Service. RETURN

Figure 2-119 FORTRAN - PL/I Interface Structure

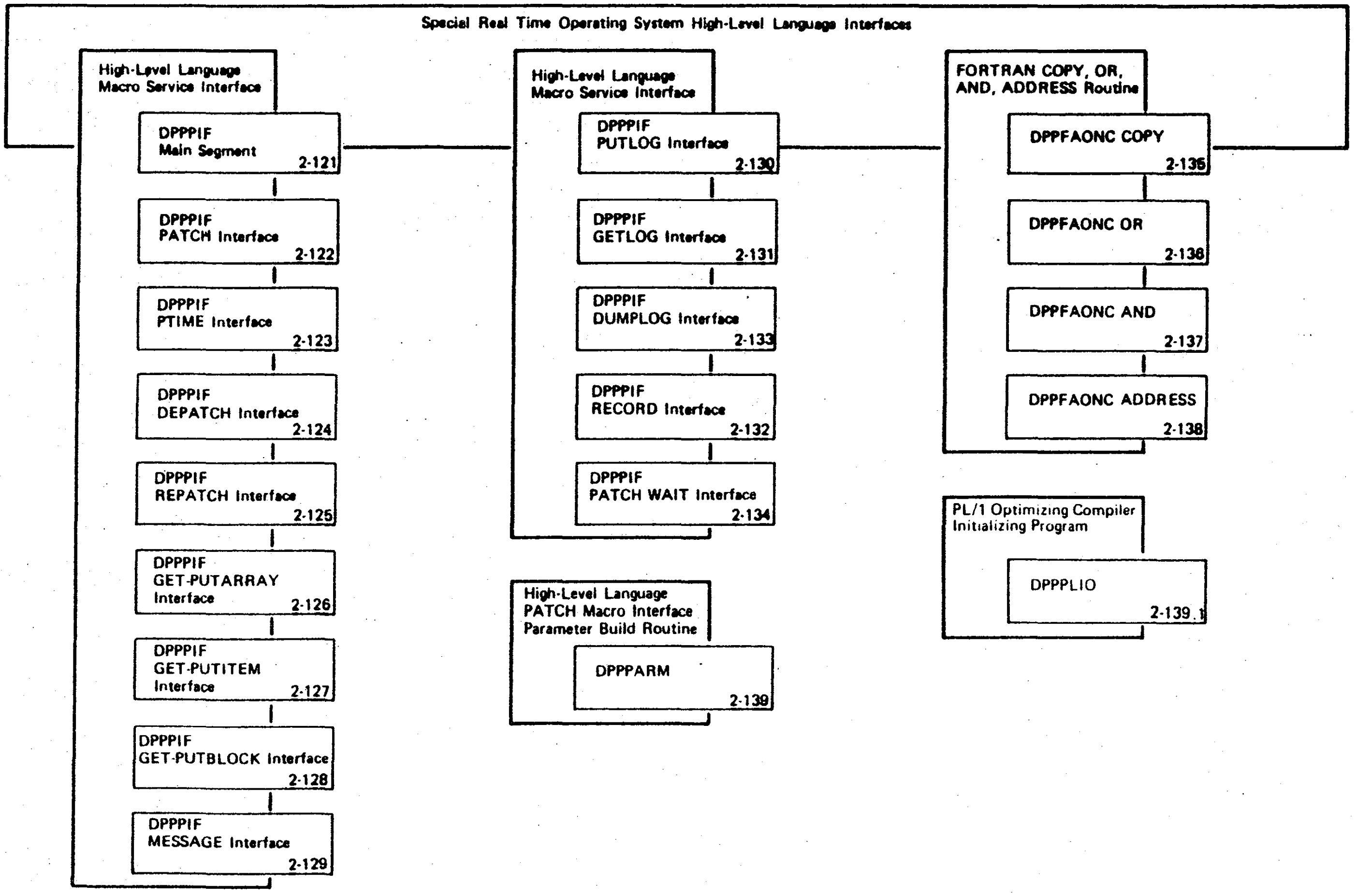

Special Real Time Operating System High-Level Language Interfaces
High-Level Language Macro Service Interface
DPPPIF Main Segment 2-121
DPPPIF PATCH Interface 2-122
DPPPIF PTIME Interface 2-123
DPPPIF DEPATCH Interface 2-124
DPPPIF REPATCH Interface 2-125
DPPPIF GET-PUTARRAY Interface 2-126
DPPPIF GET-PUTITEM Interface 2-127
DPPPIF GET-PUTBLOCK Interface 2-128
DPPPIF MESSAGE Interface 2-129
High-Level Language Macro Service Interface
DPPPIF PUTLOG Interface 2-130
DPPPIF GETLOG Interface 2-131
DPPPIF DUMPLOG Interface 2-133
DPPPIF RECORD Interface 2-132
DPPPIF PATCH WAIT Interface 2-134
High-Level Language PATCH Macro Interface Parameter Build Routine
DPPPARM 2-139
FORTRAN COPY, OR, AND, ADDRESS Routine
DPPFAONC COPY 2-135
DPPFAONC OR 2-136
DPPFAONC AND 2-137
DPPFAONC ADDRESS 2-138
PL/1 Optimizing Compiler Initializing Program
DPPPLIO 2-139.1

Figure 2-120 (1 of 2) Special Real Time Operating System High-Level Language Interface Overview

Intentionally Blank

DPPPIF

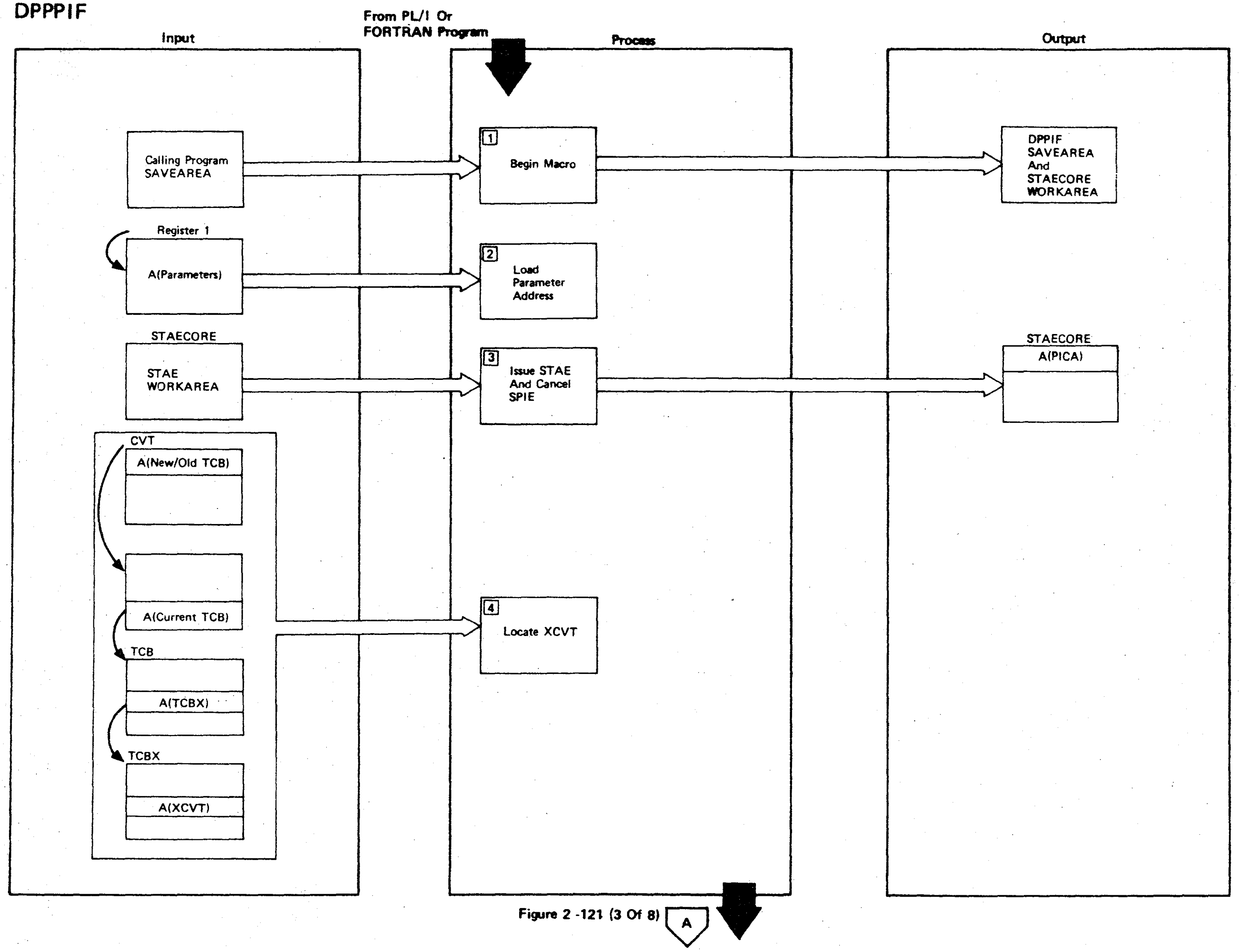


Figure 2-121 (1 Of 8) - High-Level Language Macro Service Interface Main Segment

Figure 2-121 (2 of 8).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	A combination 72 byte save area and 12 byte STAECORE work area is obtained after the caller's registers are saved in the save area pointed to by register 13. The save areas are chained together with register 13 becoming the base for the SAVEEM DSECT.		DPPPIF
2	Load parameter address.		DPPPIF
3	The high-level language SPIE exit is canceled and the address saved for the return logic. A STAE is issued to provide a means for freeing core should an ABEND occur due to an error in the user parameter list.		
4	The address of the Special Real Time Operating System XCVT will be located.		DPPPIF

DPPPIF

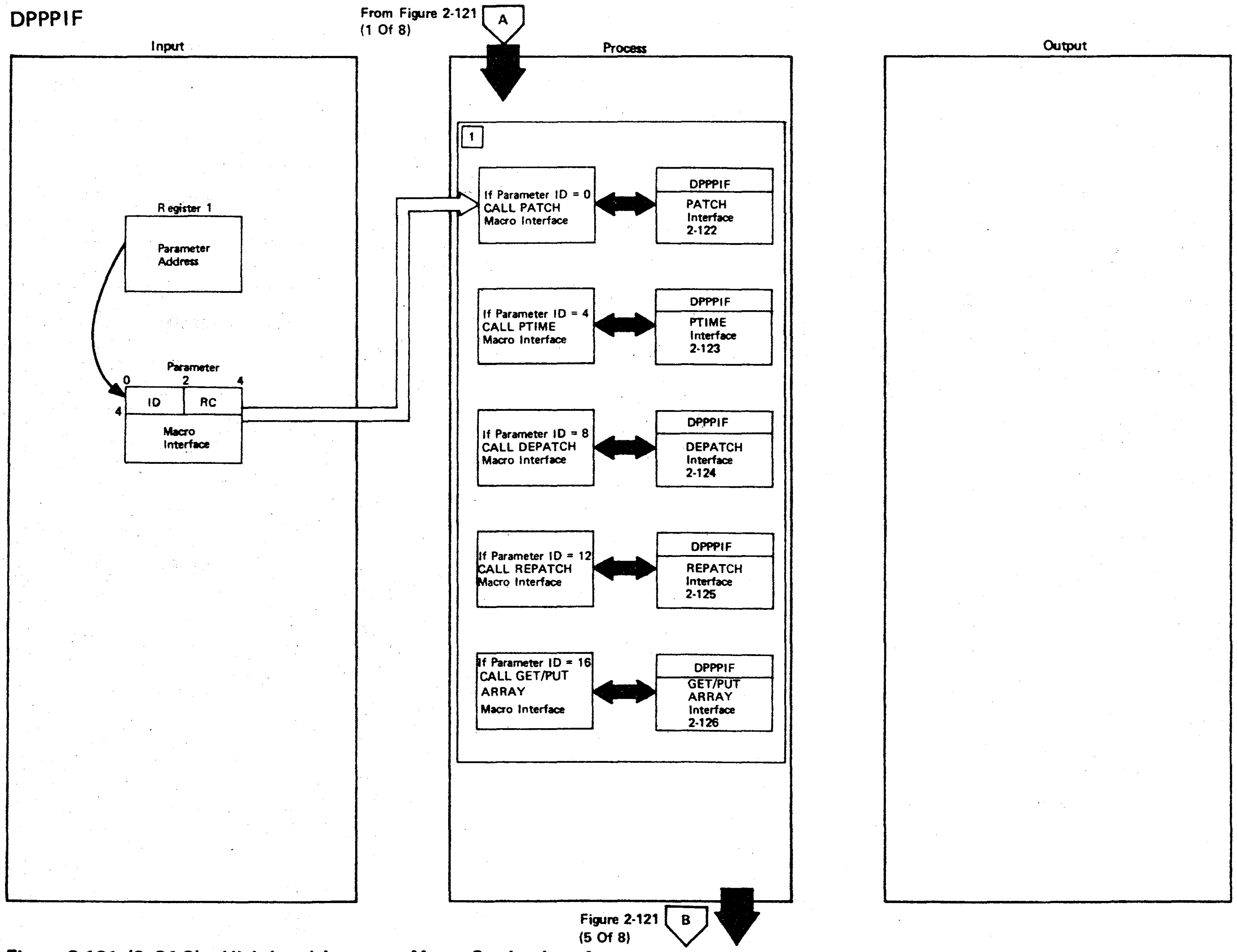

From Figure 2-121
(1 Of 8)
A
Input
Process
Output
1
Register 1
Parameter Address
Parameter
0
2
4
4
ID
RC
Macro Interface
If Parameter ID = 0 CALL PATCH Macro Interface
DPPPIF
PATCH Interface 2-122
If Parameter ID = 4 CALL PTIME Macro Interface
DPPPIF
PTIME Interface 2-123
If Parameter ID = 8 CALL DEPATCH Macro Interface
DPPPIF
DEPATCH Interface 2-124
If Parameter ID = 12 CALL REPATCH Macro Interface
DPPPIF
REPATCH Interface 2-125
If Parameter ID = 16 CALL GET/PUT ARRAY Macro Interface
DPPPIF
GET/PUT ARRAY Interface 2-126
Figure 2-121
(5 Of 8)
B

Figure 2-121 (3 Of 8) - High-Level Language Macro Service Interface

Figure 2-121 (4 of 8).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Based on the value of the ID, one of the following interface segments in DPPPIF will be entered: ID Interface Segments 0 - PATCH 4 - PTIME 8 - DEPATCH 12 - REPATCH 16 - GET/PUT ARRAY		DPPPIF

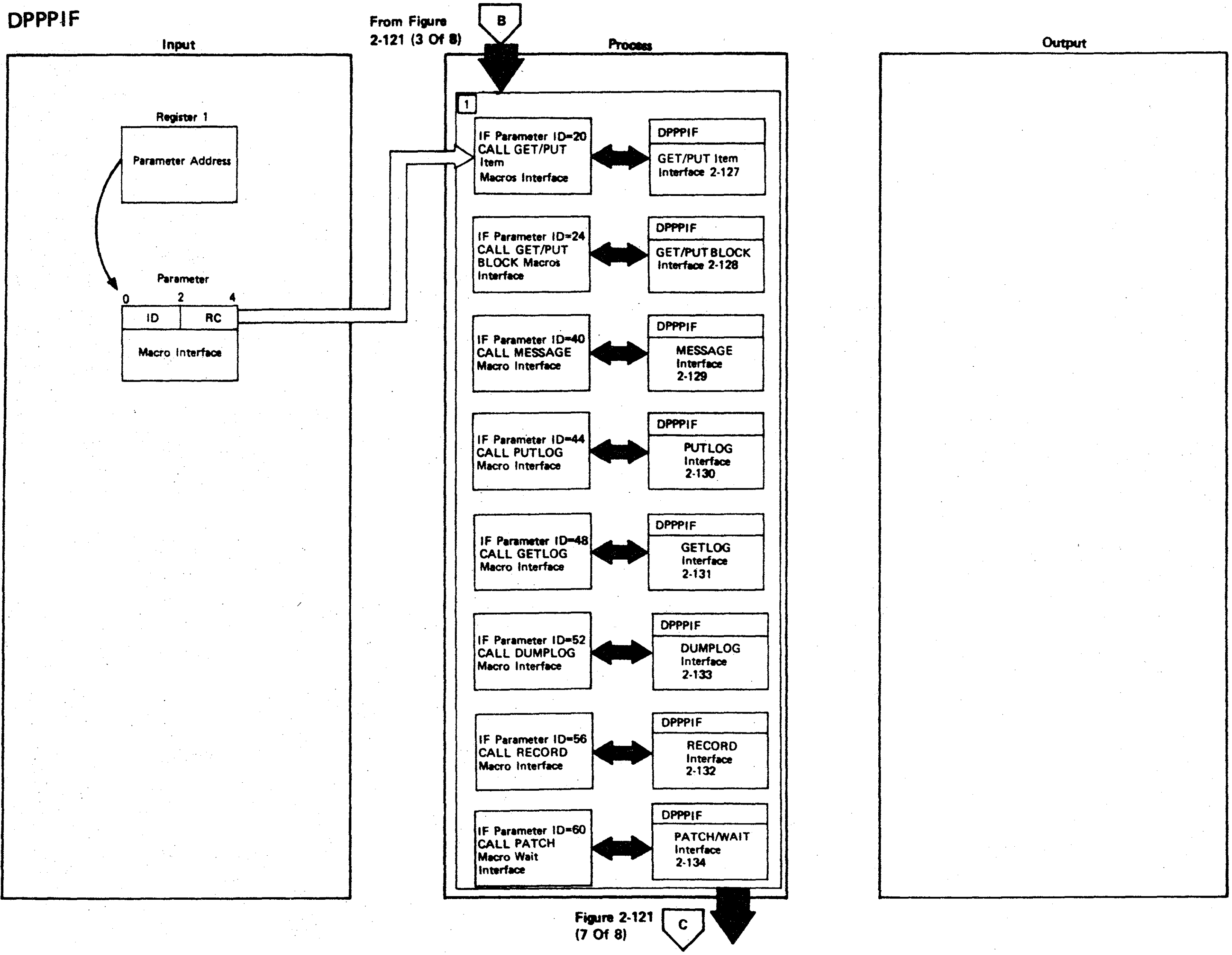

Input
Register 1
Parameter Address
Parameter
0
2
4
ID
RC
Macro Interface
From Figure 2-121 (3 Of 8)
B
Process
1
IF Parameter ID=20 CALL GET/PUT Item Macros Interface
DPPPIF
GET/PUT Item Interface 2-127
IF Parameter ID=24 CALL GET/PUT BLOCK Macros Interface
DPPPIF
GET/PUT BLOCK Interface 2-128
IF Parameter ID=40 CALL MESSAGE Macro Interface
DPPPIF
MESSAGE Interface 2-129
IF Parameter ID=44 CALL PUTLOG Macro Interface
DPPPIF
PUTLOG Interface 2-130
IF Parameter ID=48 CALL GETLOG Macro Interface
DPPPIF
GETLOG Interface 2-131
IF Parameter ID=52 CALL DUMPLOG Macro Interface
DPPPIF
DUMPLOG Interface 2-133
IF Parameter ID=56 CALL RECORD Macro Interface
DPPPIF
RECORD Interface 2-132
IF Parameter ID=60 CALL PATCH Macro Wait Interface
DPPPIF
PATCH/WAIT Interface 2-134
Figure 2-121 (7 Of 8)
C
Output

Figure 2-121 (5 Of 8) - High-Level Language Macro Service Interface Main Segment

Figure 2-121 (6 of 8).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Based on the value of the ID, one of the following interface segments in DPPPIF will be entered: ID Interface Segment 20 - GET/PUT ITEM 24 - GET/PUT BLOCK 40 - MESSAGE 44 - PUTLOG 48 - GETLOG 52 - DUMPLOG 56 - RECORD 60 - PATCH/WAIT		

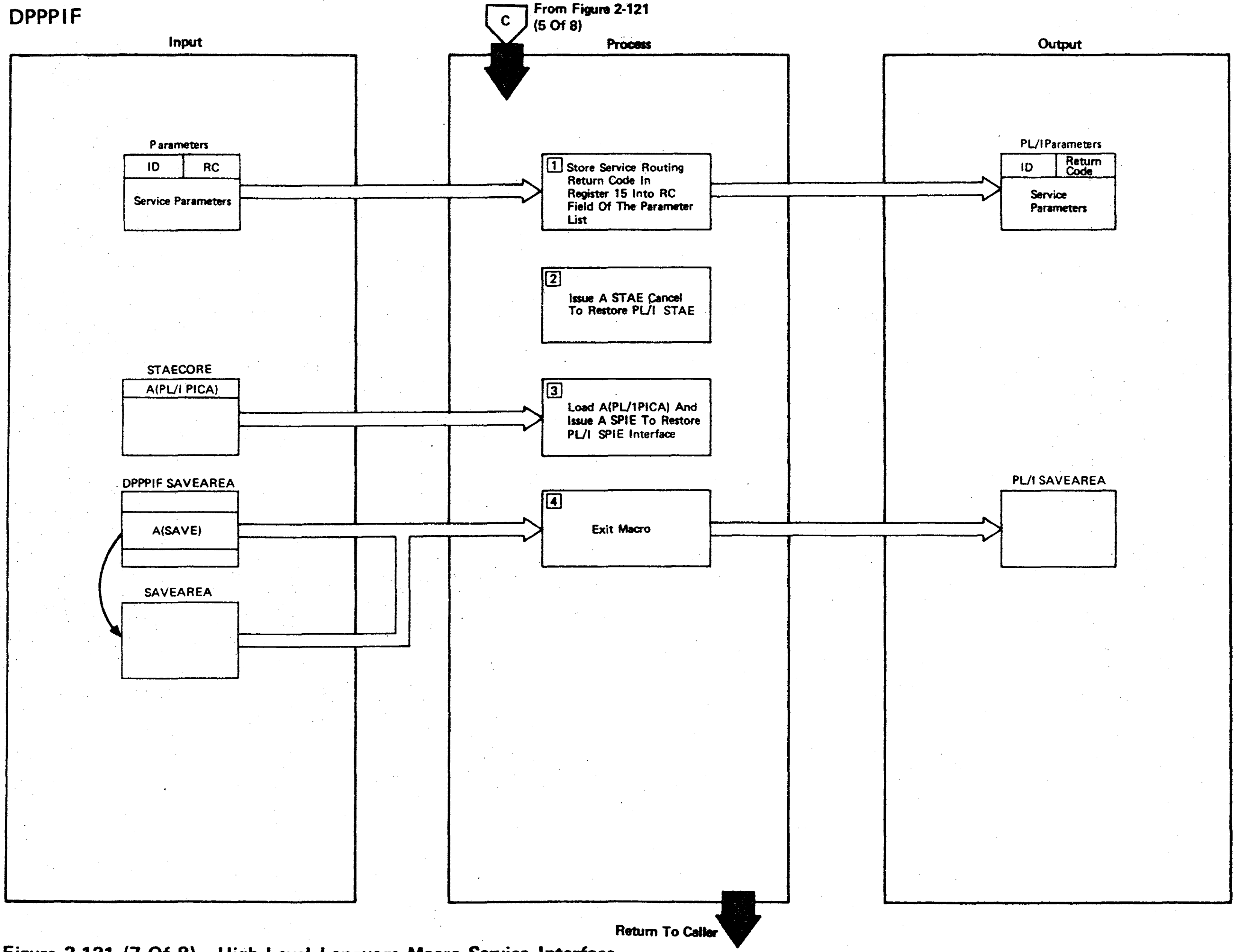

C
From Figure 2-121 (5 Of 8)
Input
Process
Output
Parameters
ID
RC
Service Parameters
1 Store Service Routing Return Code In Register 15 Into RC Field Of The Parameter List
PL/I Parameters
ID
Return Code
Service Parameters
2 Issue A STAE Cancel To Restore PL/I STAE
STAECORE
A(PL/I PICA)
3 Load A(PL/1PICA) And Issue A SPIE To Restore PL/I SPIE Interface
DPPPIF SAVEAREA
A(SAVE)
SAVEAREA
4 Exit Macro
PL/I SAVEAREA
Return To Caller

Figure 2-121 (7 Of 8) - High-Level Language Macro Service Interface

Figure 2-121 (8 of 8).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The EXIT macro frees the DPPPIF save area obtained by GETMAIN, marks the save area to indicate DPPPIF is done, reloads registers, and returns to the program via register 14.		DPPPIF

DPPPIF

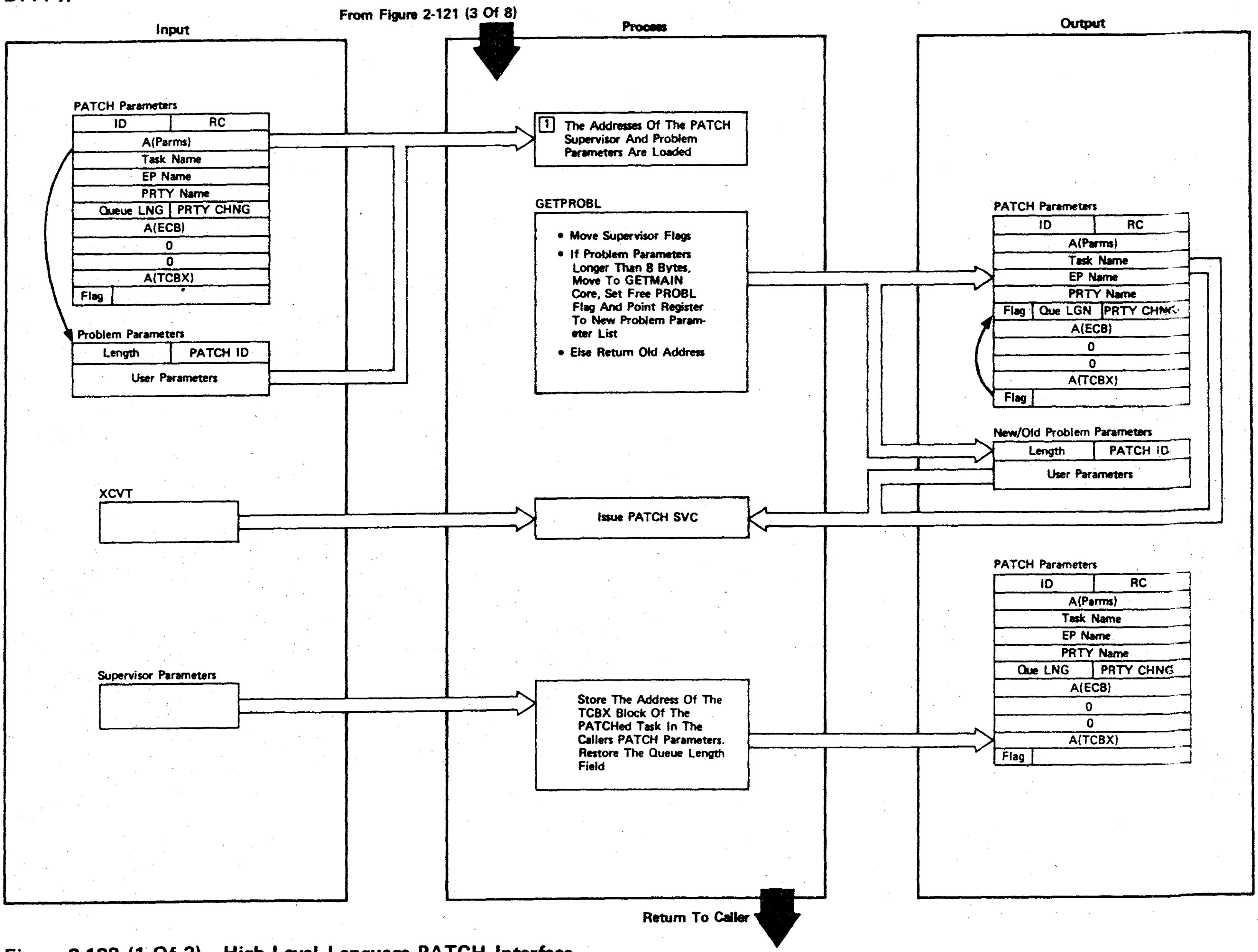


Figure 2-122 (1 Of 2) - High-Level Language PATCH Interface

Figure 2-122 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The PATCH Supervisor Flags are relocated to their correct position in the parameter list. If the problem parameters are longer than 8 bytes, they are moved to GETMAIN core, and the SUPFREEP bit is set to indicate the core is to be freed; otherwise, no move is performed. The addresses of both the old and new problem parameters are returned.		DPPPIF

DPPPIF

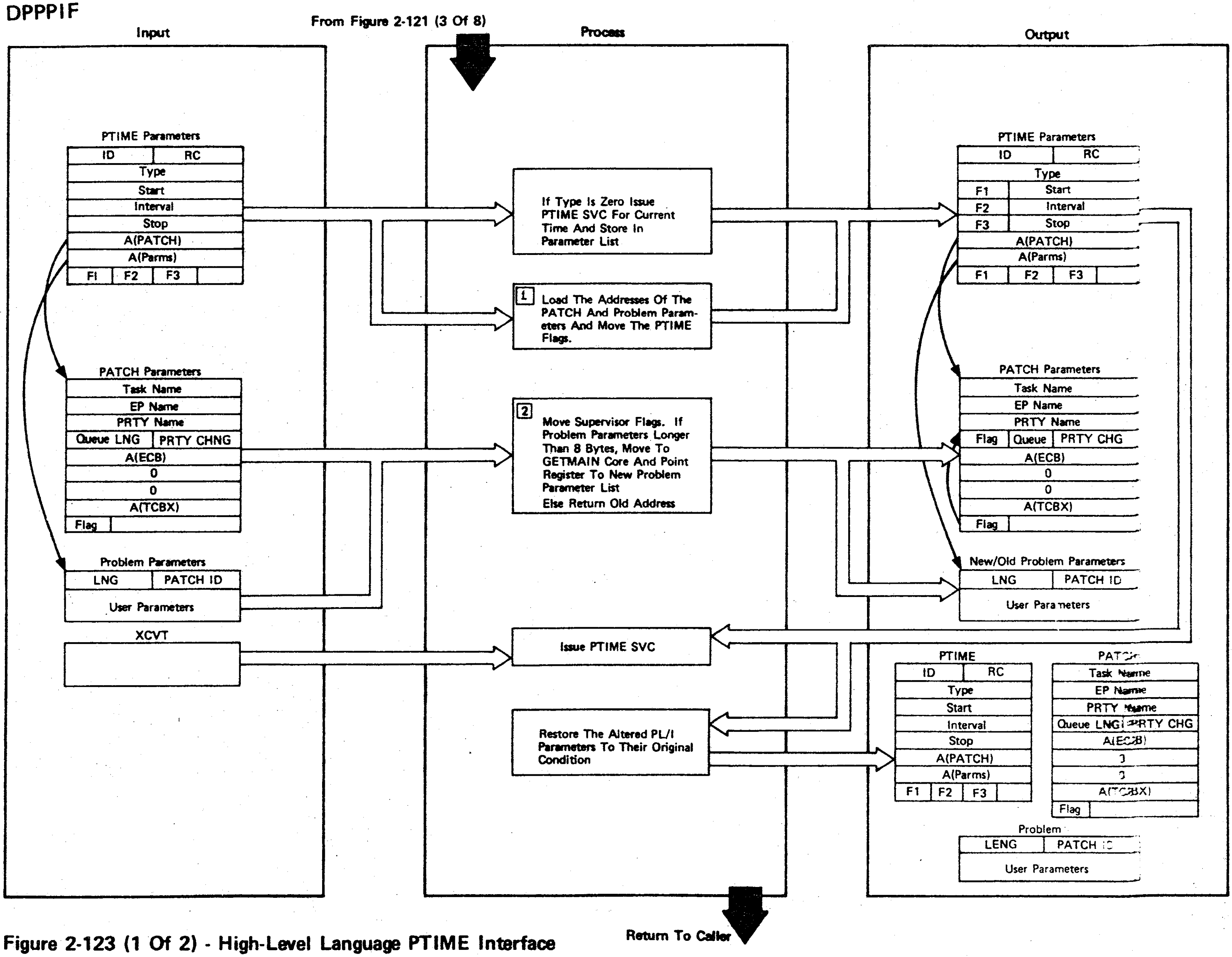


Figure 2-123 (1 Of 2) - High-Level Language PTIME Interface

Figure 2-123 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The TYPE field determines the kind of PTIME SVC and the structure of the parameter list. For TYPE=0, a PTIME TYPE=RET is issued to obtain the current time of day and the address of the Special Real Time Operating System time array.		DPPPIF
2	The PATCH Supervisor Flags are relocated to their correct position in the parameter list. If the problem parameters are longer than 8 bytes, they are moved to GETMAIN core, and the SUPFREEP bit is set to indicate the core is to be freed; otherwise, no move is performed. The addresses of both the old and new problem parameters are returned.		DPPPIF

DPPPIF

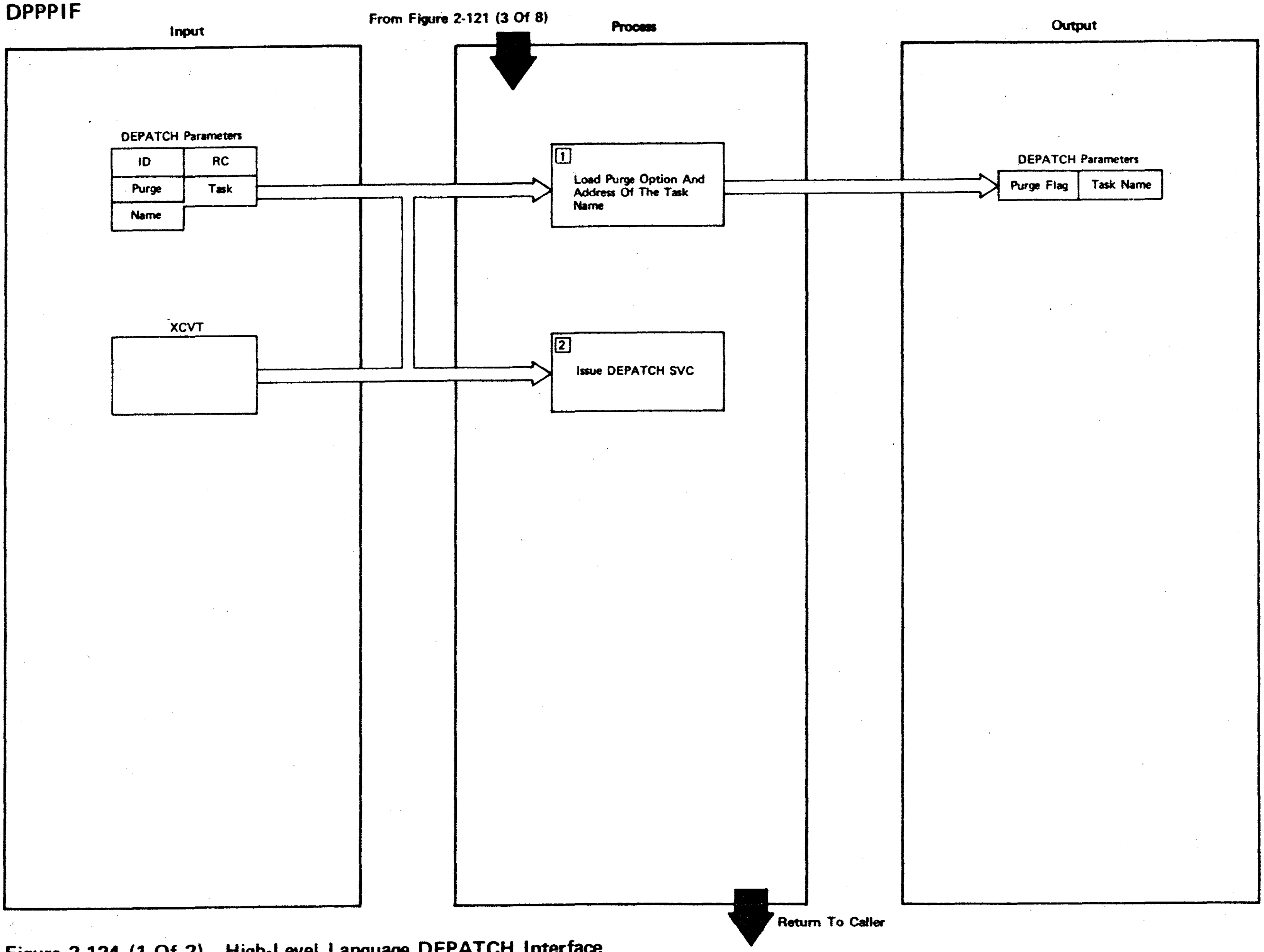


Figure 2-124 (1 Of 2) - High-Level Language DEPATCH Interface

Figure 2-124 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Load the DEPATCH purge option and the address of the task name.		DPPPIF
2	Issue a DEPATCH SVC with passed DEPATCH parameters.		DPPPIF

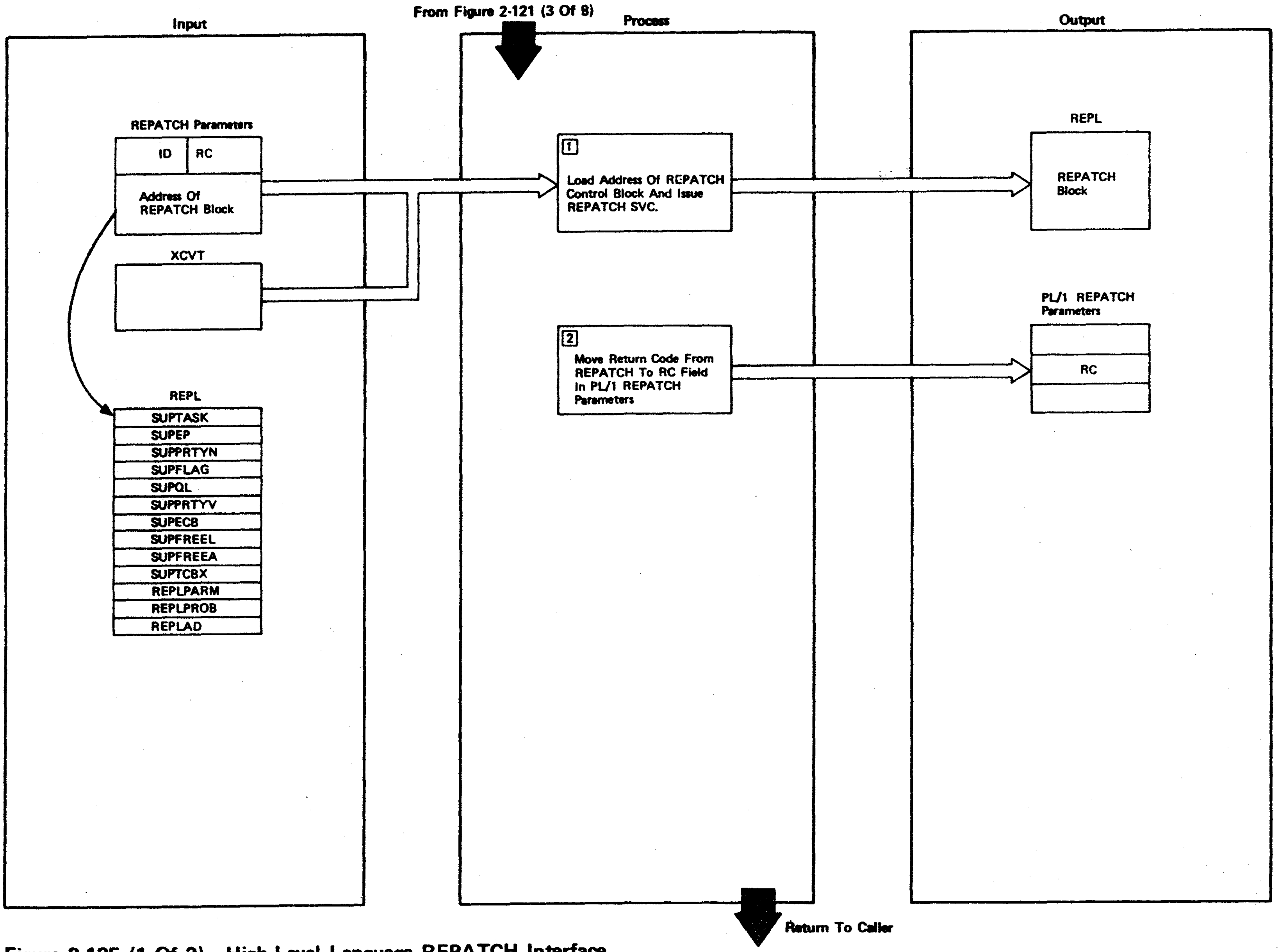


Figure 2-125 (1 Of 2) - High-Level Language REPATCH Interface

Figure 2-125 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Load the address of the REPATCH control block (REPL) and issue REPATCH SVC.		DPPPIF
2	The return code in register 15 (REPATCH return code) will be stored in the REPATCH parameters.		DPPPIF

DPPPIF

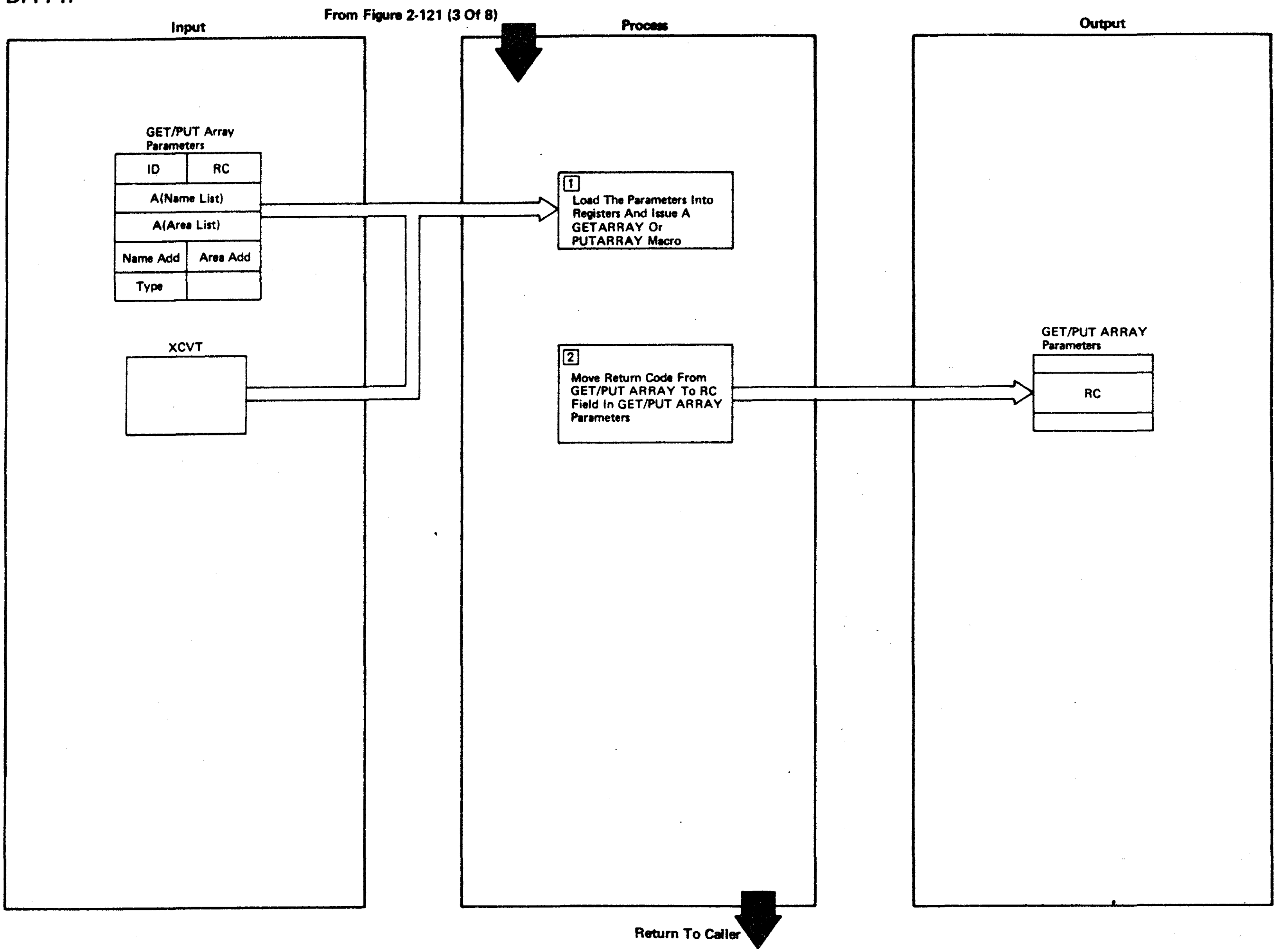

From Figure 2-121 (3 Of 8)
Input
Process
Output
GET/PUT Array Parameters
ID
RC
A(Name List)
A(Area List)
Name Add
Area Add
Type
XCVT
1 Load The Parameters Into Registers And Issue A GETARRAY Or PUTARRAY Macro
2 Move Return Code From GET/PUT ARRAY To RC Field In GET/PUT ARRAY Parameters
GET/PUT ARRAY Parameters
RC
Return To Caller

Figure 2-126 (1 Of 2) - High-Level Language GET/PUTARRAY Interface

Figure 2-126 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Load the address of GETARRAY/PUTARRAY parameters and issue the GETARRAY-PUTARRAY macro.		DPPPIF
2	The return code in register 15 (GETARRAY/PUTARRAY return code) will be stored in the GETARRAY/PUTARRAY parameters.		DPPPIF

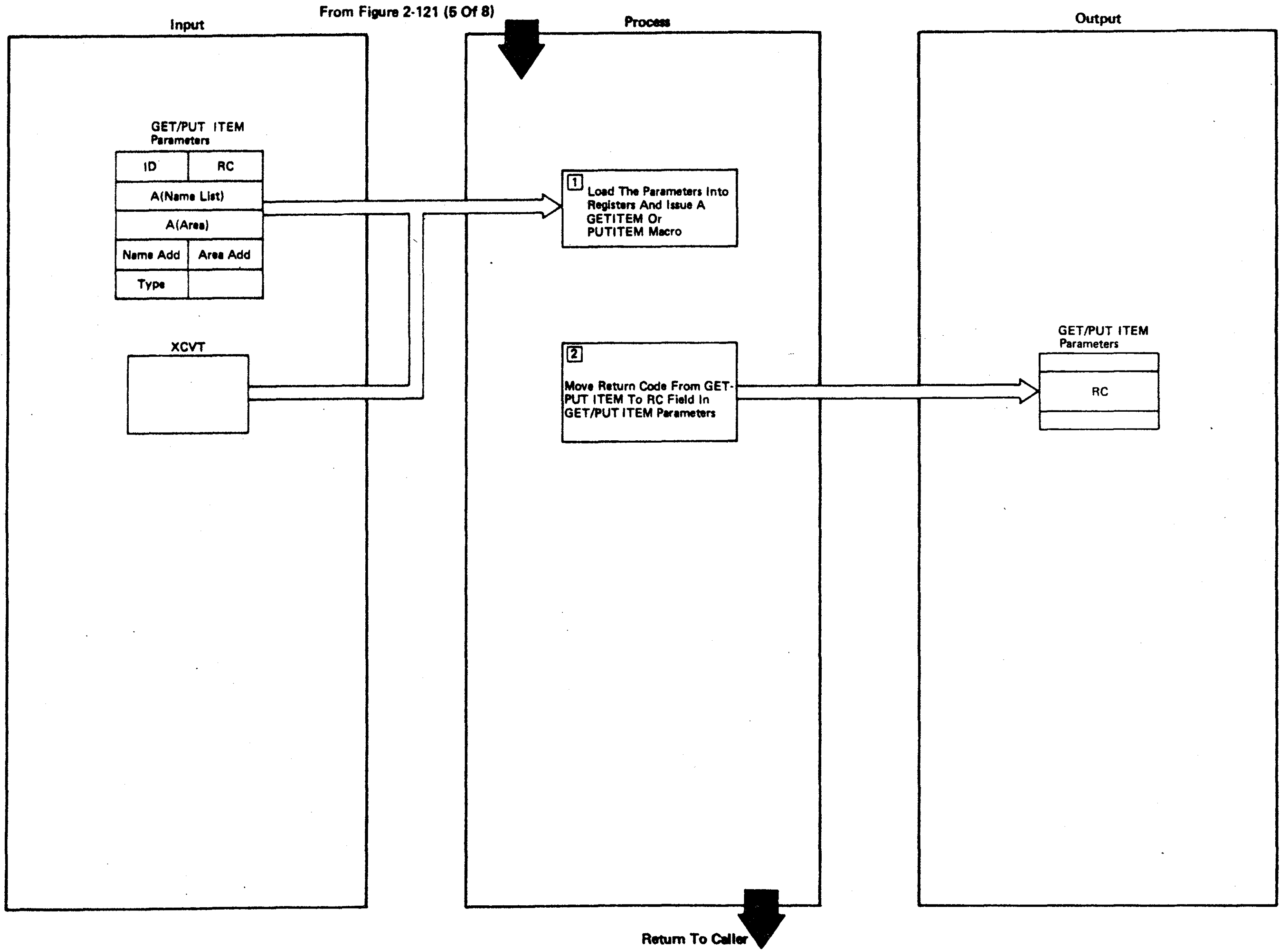

From Figure 2-121 (5 Of 8)
Input
Process
Output
GET/PUT ITEM Parameters
ID
RC
A(Name List)
A(Area)
Name Add
Area Add
Type
XCVT
1 Load The Parameters Into Registers And Issue A GETITEM Or PUTITEM Macro
2 Move Return Code From GET-PUT ITEM To RC Field In GET/PUT ITEM Parameters
GET/PUT ITEM Parameters
RC
Return To Caller

Figure 2-127 (1 Of 2) - High-Level Language GET/PUTITEM Interface

Figure 2-127 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Load the address of GETITEM/PUTITEM parameters and issue a GETITEM/PUTITEM macro.		DPPPIF
2	The return code in register 15 (GETITEM/PUTITEM return code) will be stored in the GETITEM/PUTITEM parameters.		DPPPIF

DPPPIF

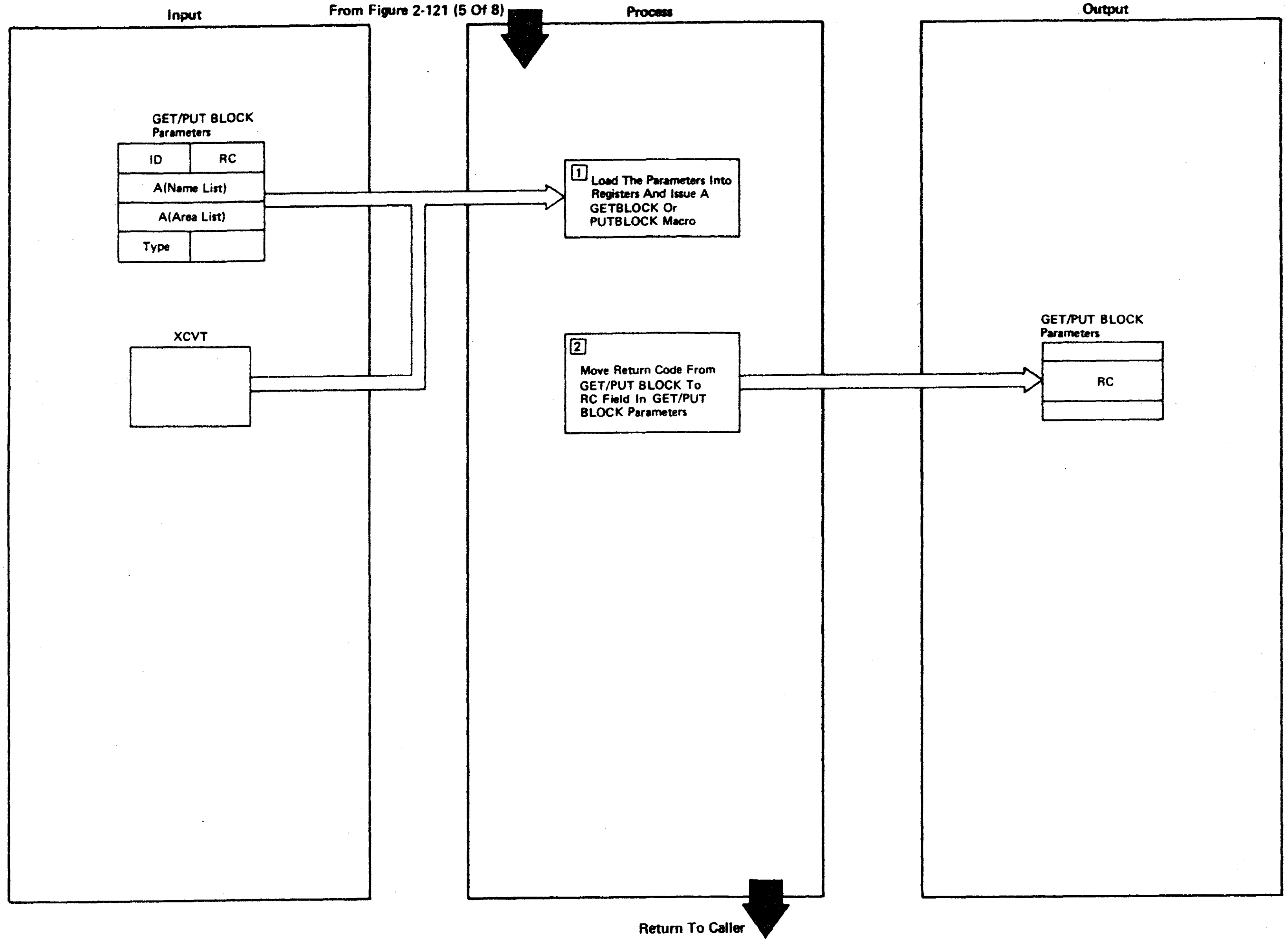

Input
From Figure 2-121 (5 Of 8)
Process
Output
GET/PUT BLOCK Parameters
ID
RC
A(Name List)
A(Area List)
Type
XCVT
1 Load The Parameters Into Registers And Issue A GETBLOCK Or PUTBLOCK Macro
2 Move Return Code From GET/PUT BLOCK To RC Field In GET/PUT BLOCK Parameters
GET/PUT BLOCK Parameters
RC
Return To Caller

Figure 2-128 (1 Of 2) - High-Level Language GET/PUTBLOCK Interface

Figure 2-128 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Load the address of GETBLOCK/PUTBLOCK parameters and issue a GETBLOCK macro.		DPPPIF
2	The return code in register 15 (GETBLOCK/PUTBLOCK return code) will be stored in the GETBLOCK/PUTBLOCK parameters.		DPPPIF

DPPPIF

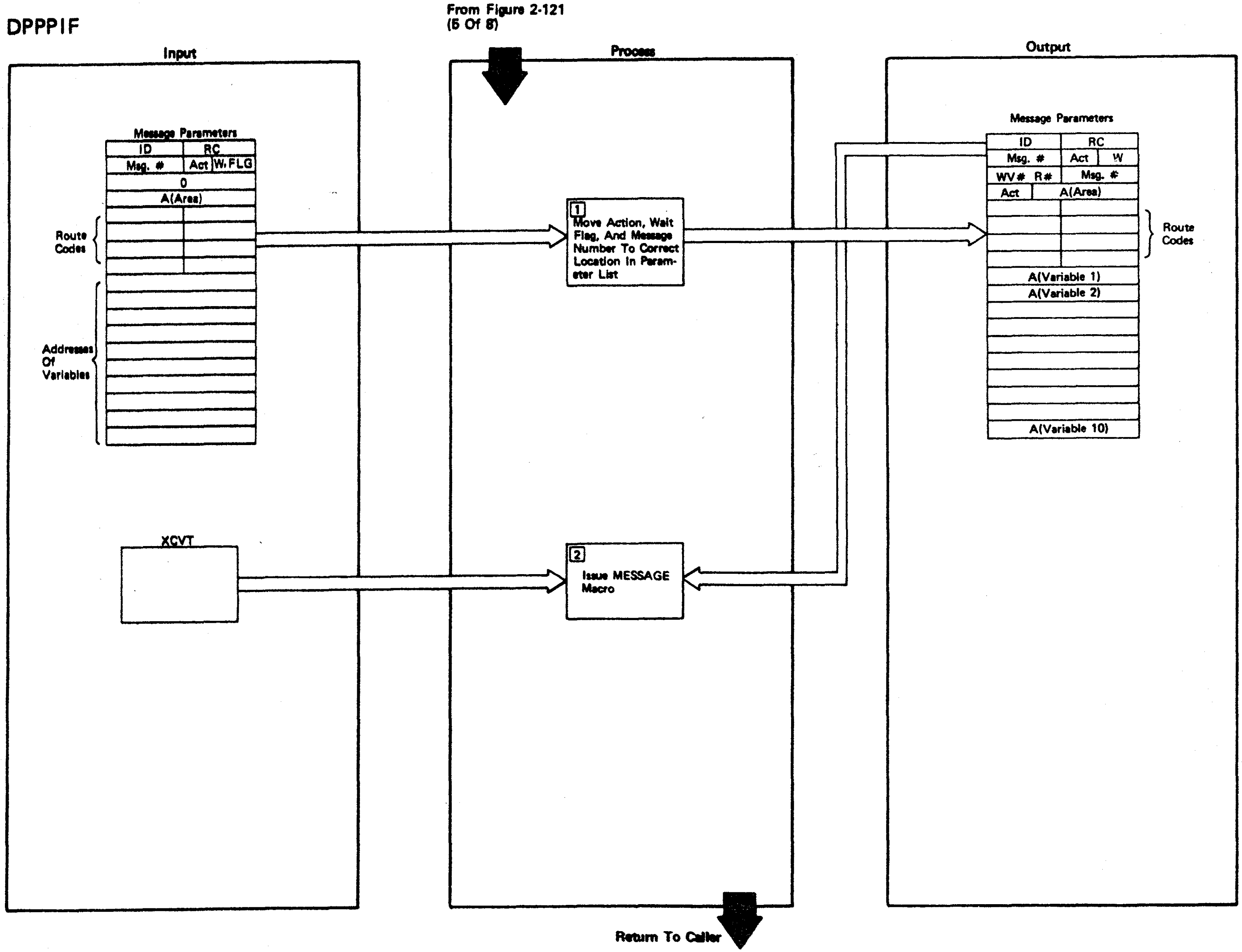


Figure 2-129 (1 Of 2) - High-Level Language Message Interface

Figure 2-129 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Move action, wait flag, and message number to correct location in the message parameter.		DPPPIF
2	Issue a MESSAGE macro.		DPPPIF

DPPPIF

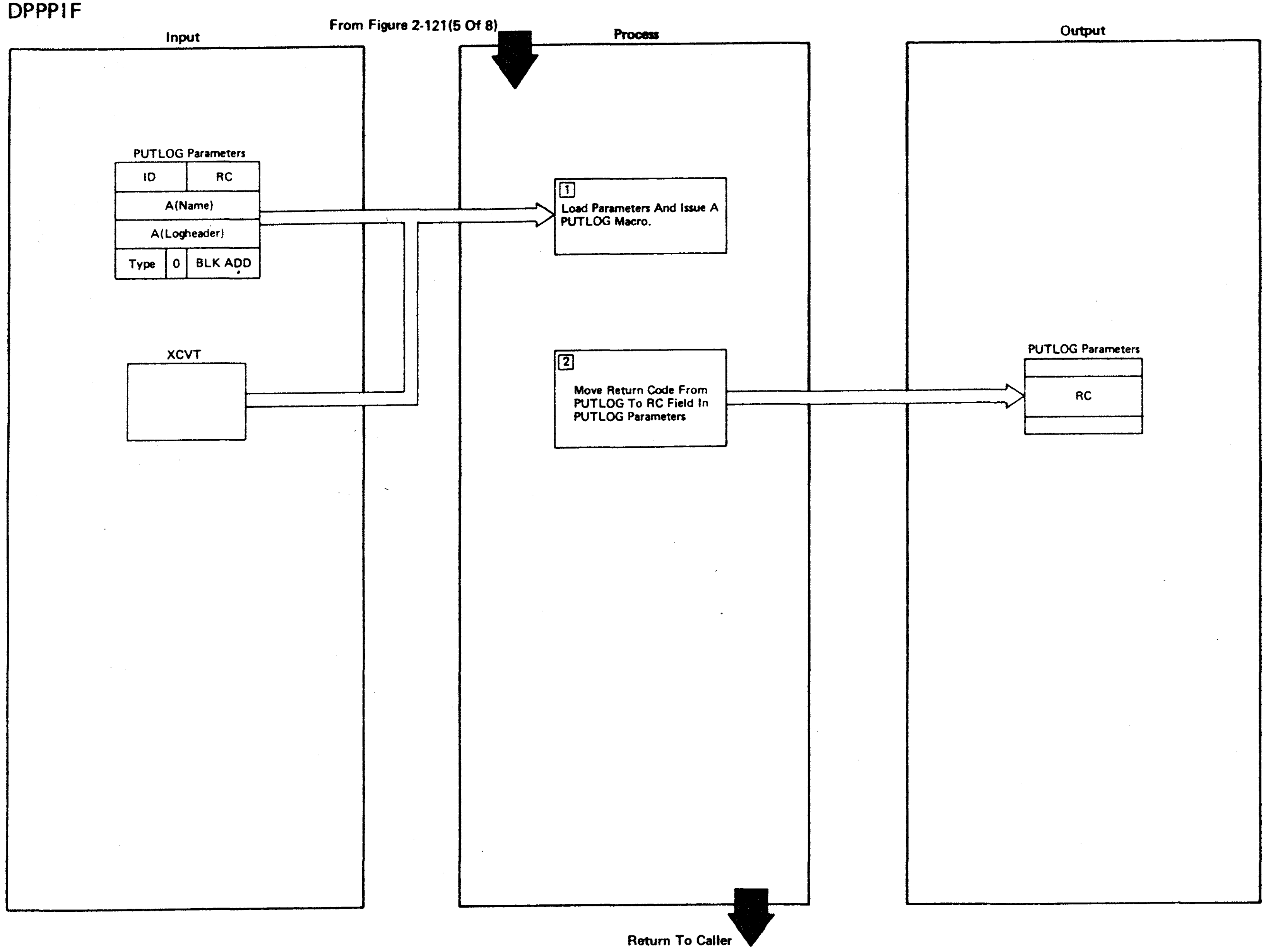

From Figure 2-121(5 Of 8)
Input
Process
Output
PUTLOG Parameters
ID
RC
A(Name)
A(Logheader)
Type
0
BLK ADD
XCVT
1
Load Parameters And Issue A PUTLOG Macro.
2
Move Return Code From PUTLOG To RC Field In PUTLOG Parameters
PUTLOG Parameters
RC
Return To Caller

Figure 2-130 (1 Of 2) - High-Level Language PUTLOG Interface

Figure 2-130 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Load parameters and issue a PUTLOG macro.		DPPPIF
2	The return code in register 15 (PUTLOG return code) will be stored in the PUTLOG parameters.		DPPPIF

DPPPIF

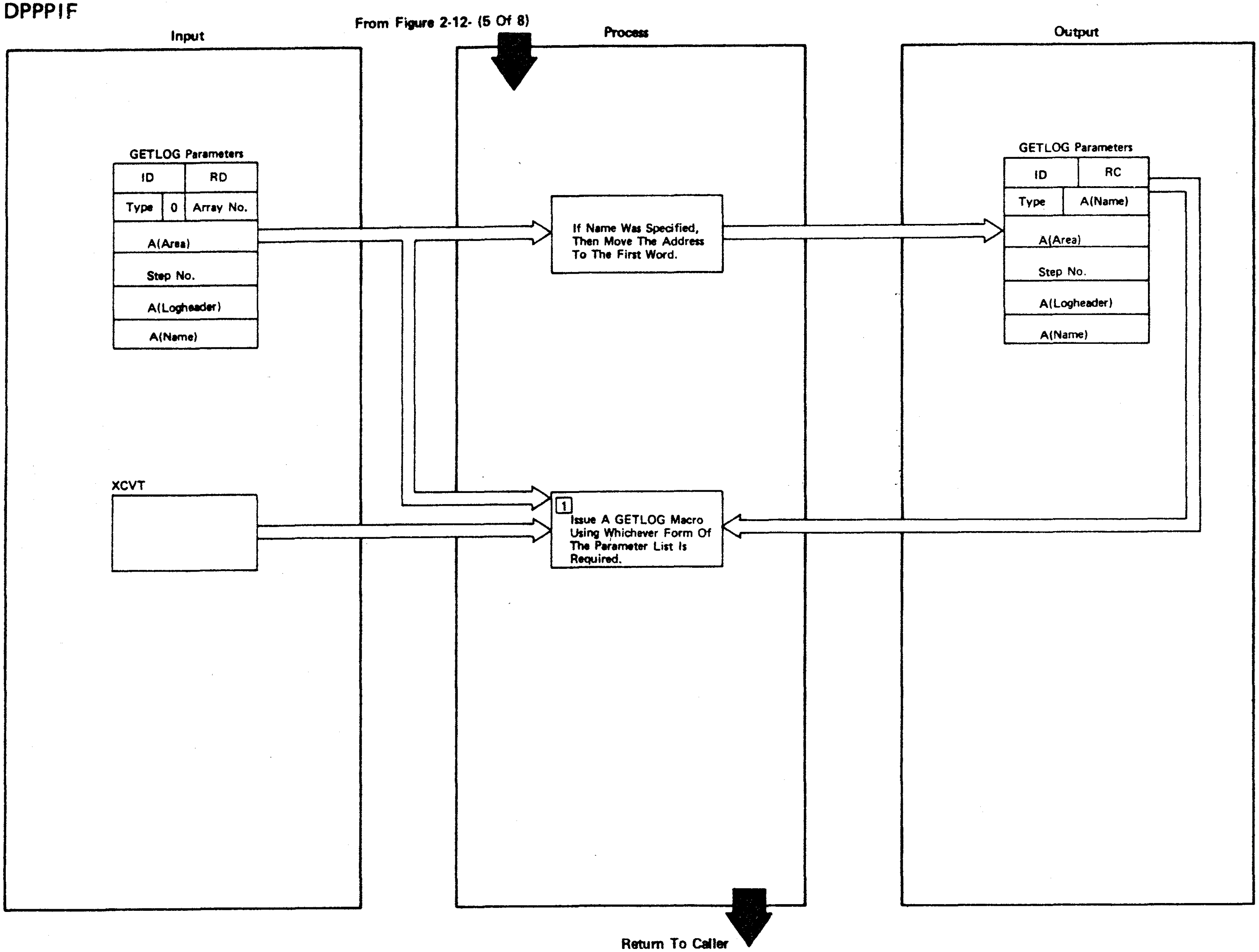


Figure 2-131 (1 Of 2) - High-Level Language GETLOG Interface

Figure 2-131 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Load parameters and issue a GETLOG macro.		DPPPIF

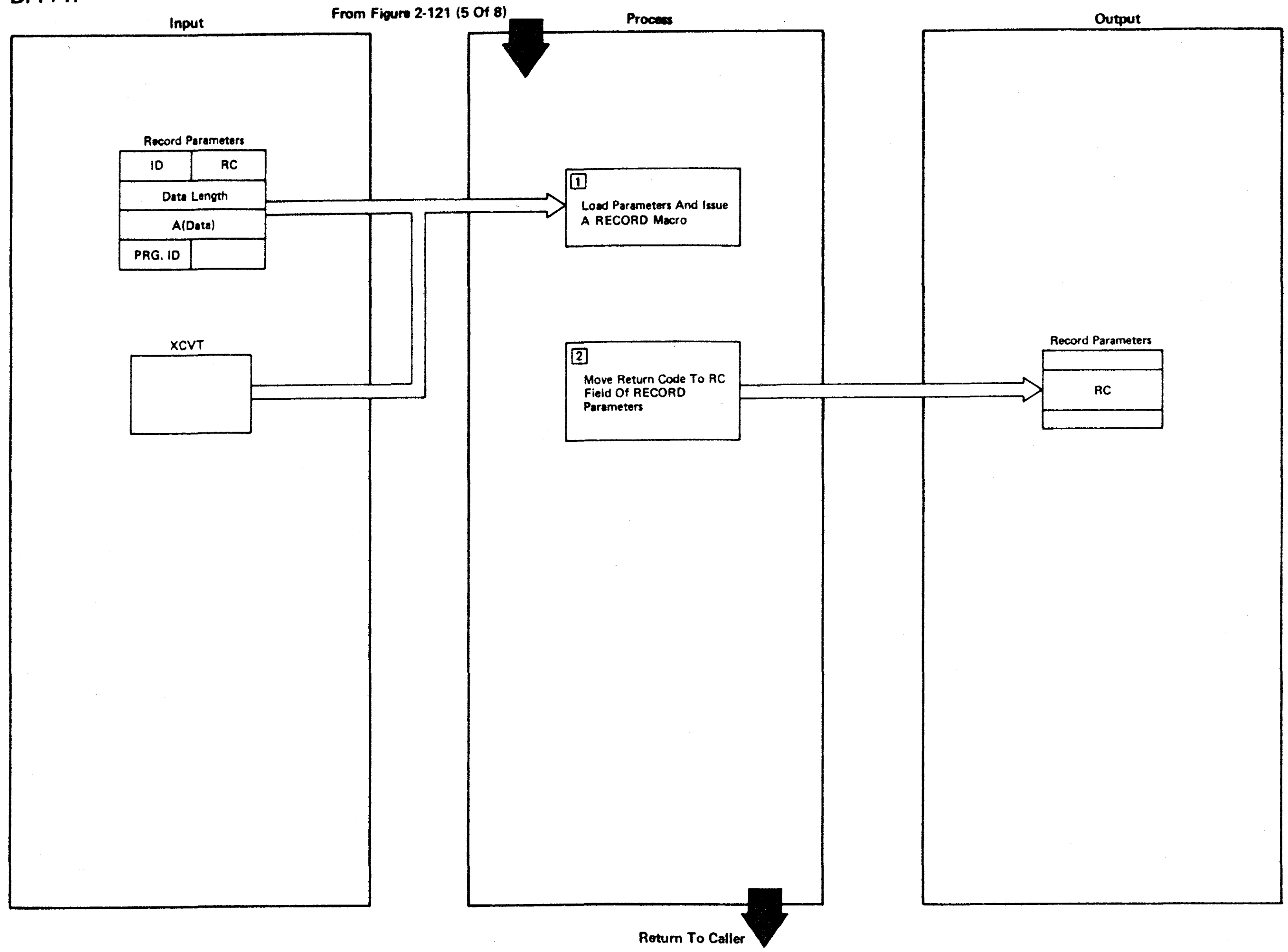

Input
From Figure 2-121 (5 Of 8)
Process
Output
Record Parameters
ID
RC
Data Length
A(Data)
PRG. ID
XCVT
1
Load Parameters And Issue A RECORD Macro
2
Move Return Code To RC Field Of RECORD Parameters
Record Parameters
RC
Return To Caller

Figure 2-132 (1 Of 2) - High-Level Language Record Interface

Figure 2-132 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Load parameters and issue a RECORD macro for the parameters specified.		DPPPIF
2	Move RECORD return code to the return code field of the of the RECORD parameters.		DPPPIF

LICENSED MATERIAL — PROPERTY OF IBM

DPPPIF

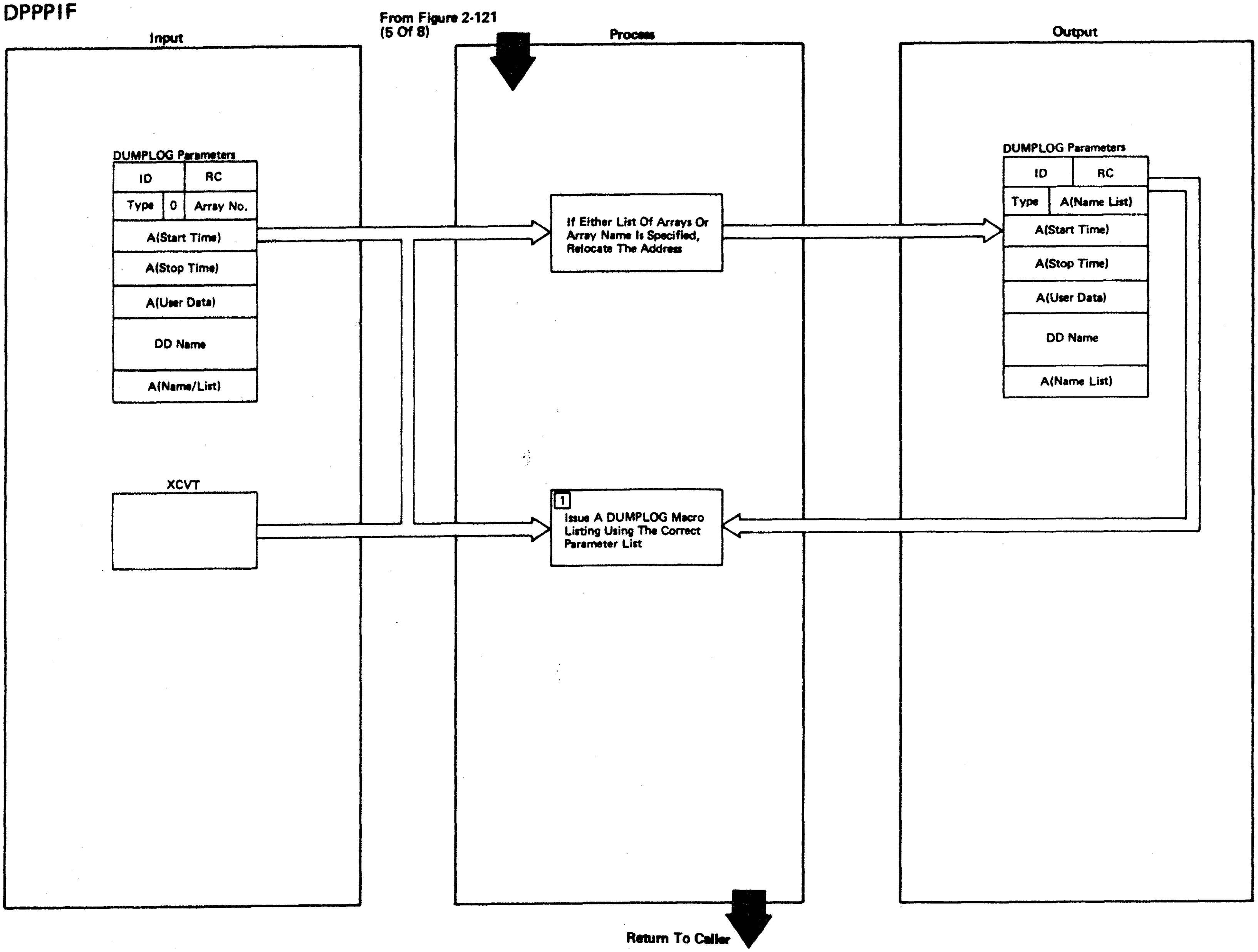


Figure 2-133 (1 Of 2) - High-Level Language DUMPLOG Interface

Figure 2-133 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Issue a DUMPLOG macro for the parameters specified.		DPPPIF

DPPPIF

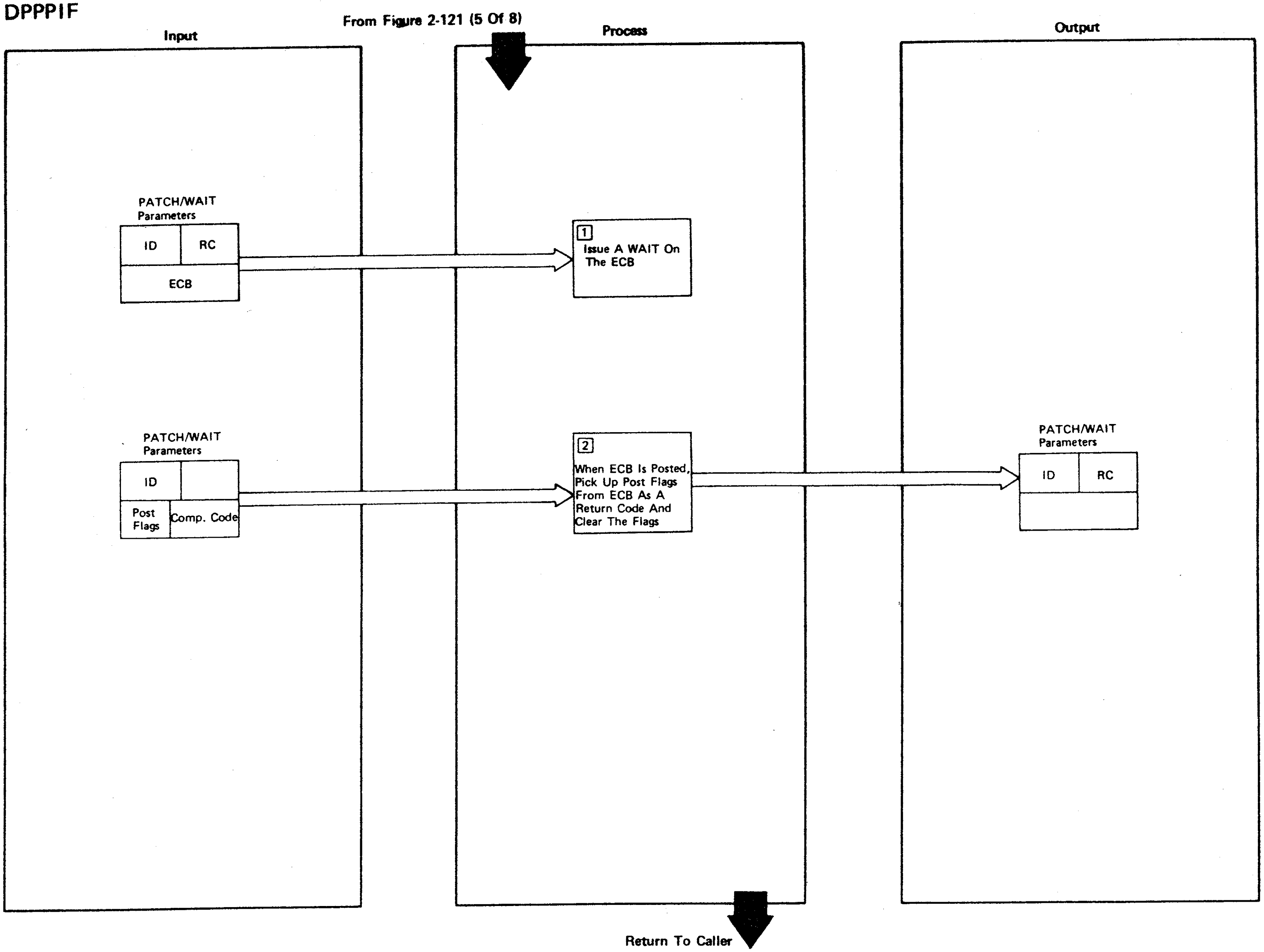

From Figure 2-121 (5 Of 8)
Input
Process
Output
PATCH/WAIT Parameters
ID
RC
ECB
1 Issue A WAIT On The ECB
PATCH/WAIT Parameters
ID
Post Flags
Comp. Code
2 When ECB Is Posted, Pick Up Post Flags From ECB As A Return Code And Clear The Flags
PATCH/WAIT Parameters
ID
RC
Return To Caller

Figure 2-134 (1 Of 2)- High-Level Language PATCH/WAIT Interface

Figure 2-134 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	An OS/VS1 WAIT macro will be issued on the ECB passed.		DPPPIF
2	When the ECB is posted, pick up the post flags from ECB as a return code, and clear the post flags.		DPPPIF

DPPFAONC Entry Point Copy

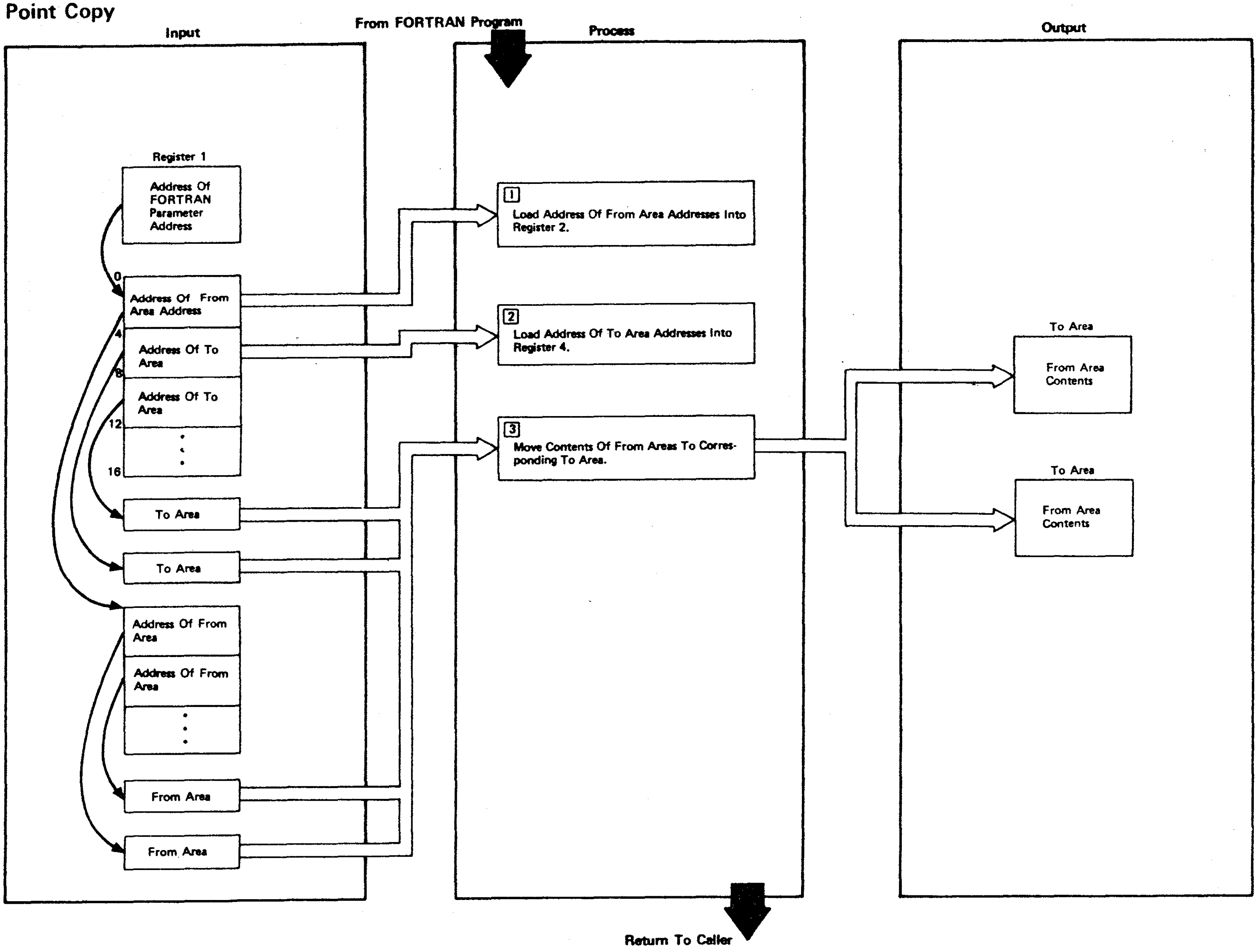


Figure 2-135 (1 Of 2)- FORTRAN Interface Copy Routine

Figure 2-135 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Load the address of 'from area addresses' into register 2.		DPPFAONC
2	Load the address of 'to area addresses' into register 4.		DPPFAONC
3	Move the contents of the 'from area' to the corresponding 'to area' contents.		DPPFAONC

DPPFAONC Entry Point ORBIT

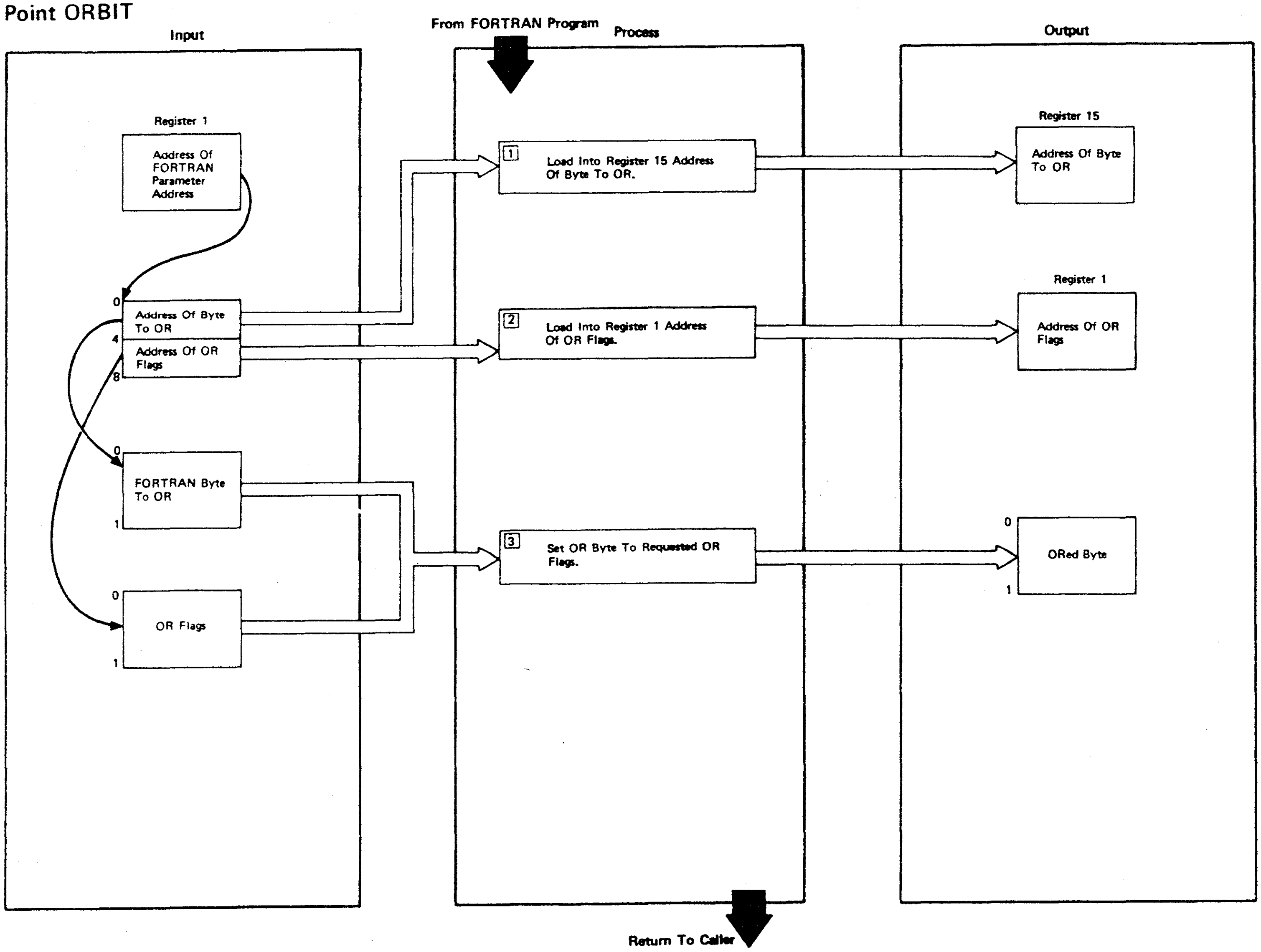


Figure 2-136 (1 Of 2) - FORTRAN Interface Or Bit (OR) Routine

Figure 2-136 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Register 15 will be loaded with the address of the byte to OR.		DPPFAONC
2	Register 1 will be loaded with the address of the OR flags.		DPPFAONC
3	The byte to be ORed will be ORed with the user supplied OR flags.		DPPFAONC

DPPFAONC Entry Point NDBIT

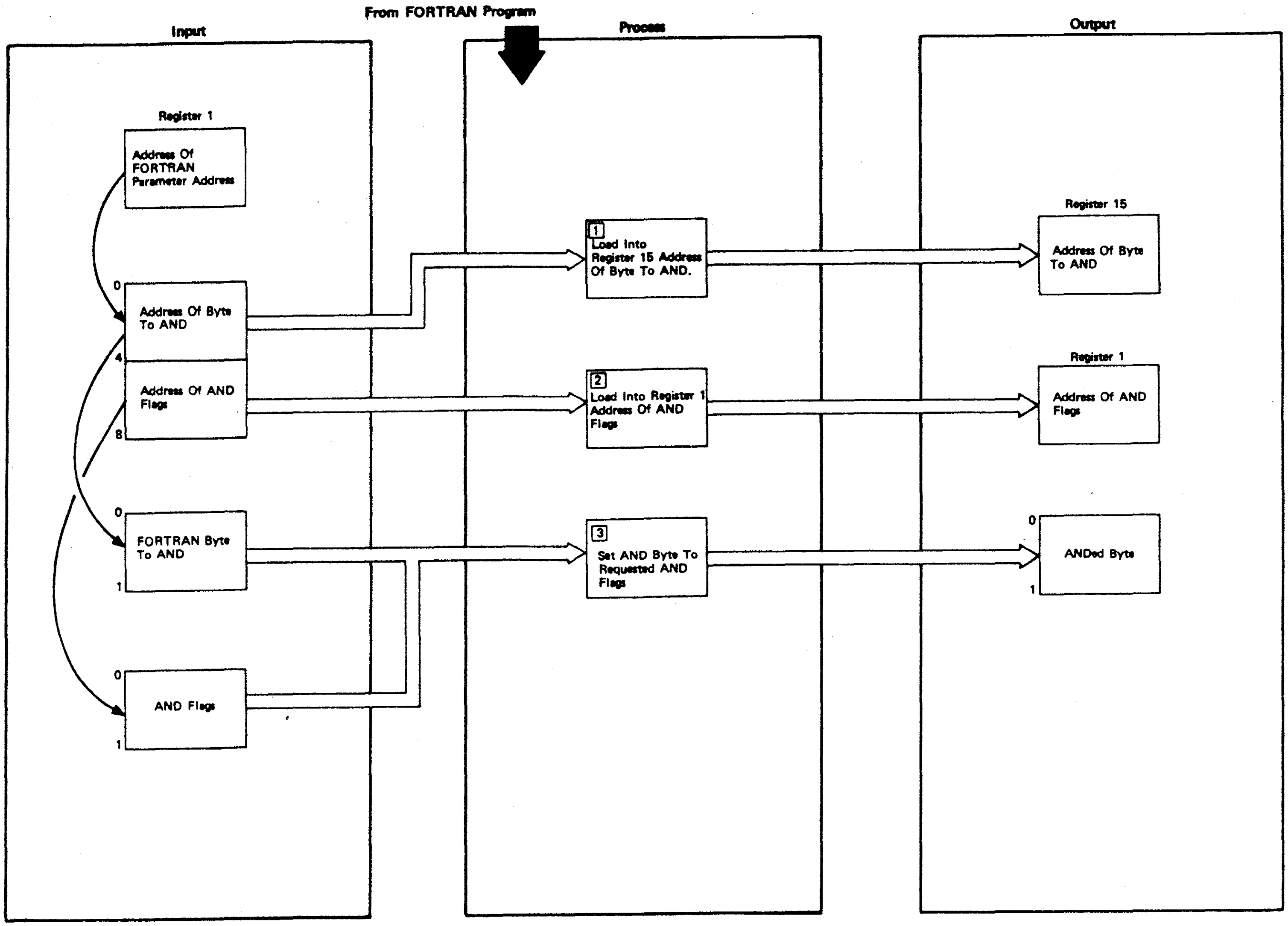


Figure 2-137 (1 Of 2) - FORTRAN Interface And Bit (AND) Routine

Figure 2-137 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Register 15 will be loaded with the address of the byte to AND.		DPPFAONC
2	Register 1 will be loaded with the address of the AND flags.		DPPFAONC
3	The byte to be ANDed will be ANDed with the user-supplied AND flags.		DPPFAONC

DPPFAONC Entry Point IADDR

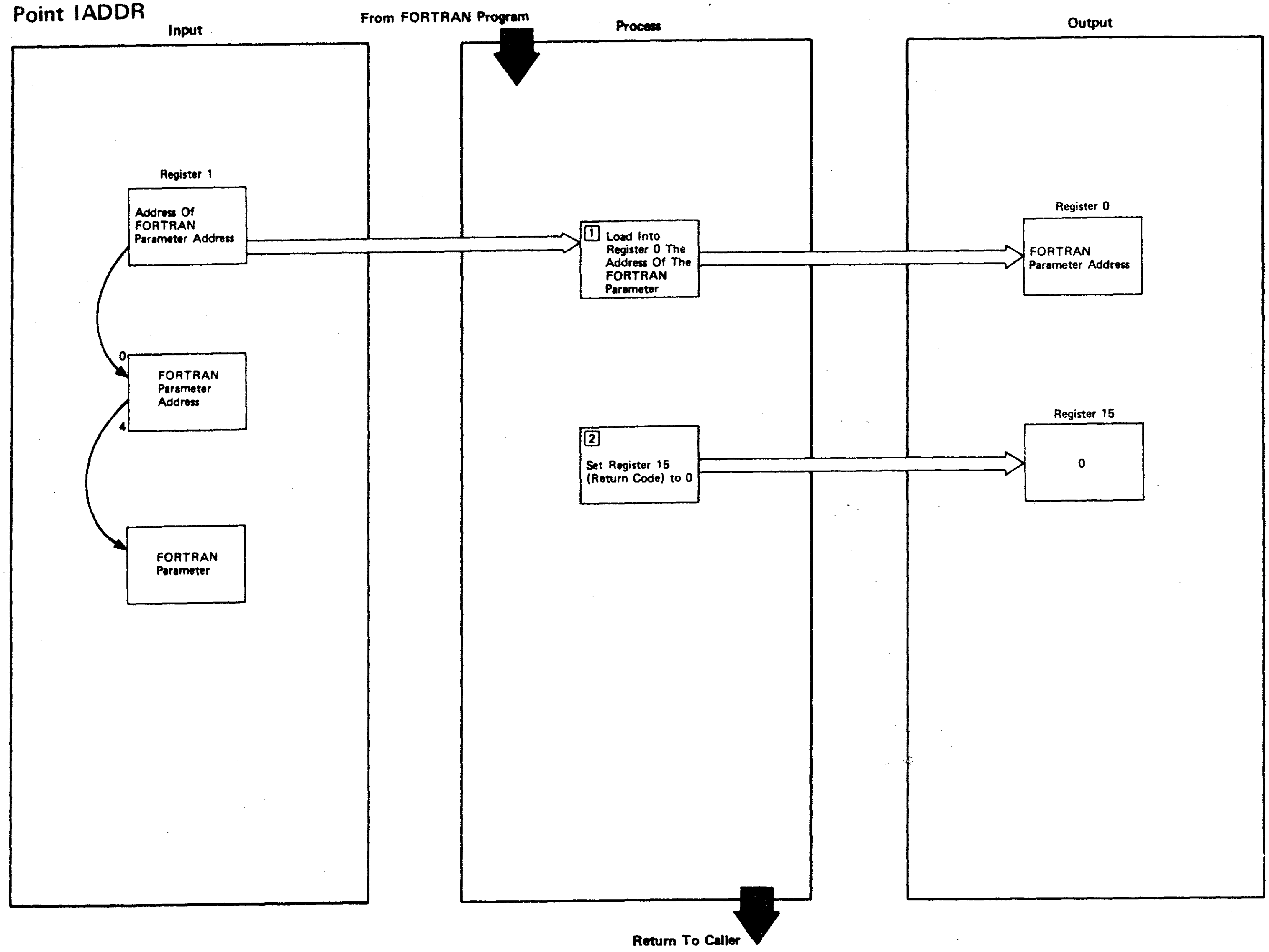

Input
From FORTRAN Program
Process
Output
Register 1
Address Of FORTRAN Parameter Address
0
FORTRAN Parameter Address
4
FORTRAN Parameter
1 Load Into Register 0 The Address Of The FORTRAN Parameter
2 Set Register 15 (Return Code) to 0
Register 0
FORTRAN Parameter Address
Register 15
0
Return To Caller

Figure 2-138 (1 Of 2) - FORTRAN Interface Address (IADDR) Routine

Figure 2-138 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The address of the FORTRAN parameter address pointed to by register 1 will be loaded into register 0.		DPPFAONC
2	On exit from the program, register 15 will be set to return code 0.		DPPFAONC

DPPPARM

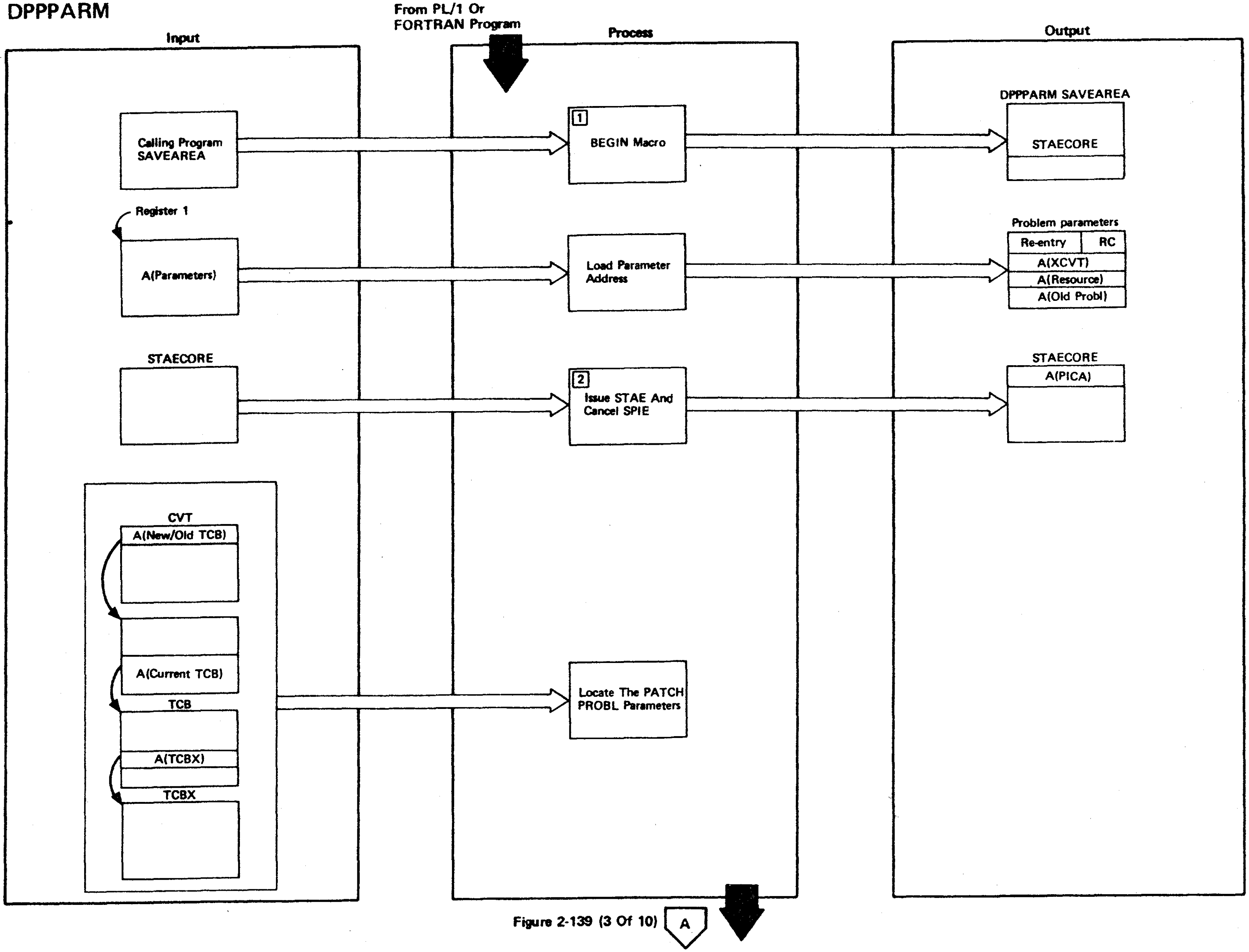


Figure 2-139 (1 Of 10) - High-Level Language PATCH Macro Interface Parameter Build Routine

Figure 2-139 (2 of 10).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	As part of the BEGIN macro, the name DPPFPM is defined as an external name with an entry point equal to the CSECT name DPPPARM. In addition, an 84 byte combination save area/work area (STAECORE) is obtained for use during execution of this routine. Register 13 is the base for this area.		DPPPARM
2	The high-level language error exits are temporarily overridden to ensure that the save area/work area obtained above is freed. The STAE routine forces a recovery to permit the subsequent entry of the high-level language STAE routine.		DPPPARM

DPPPARM

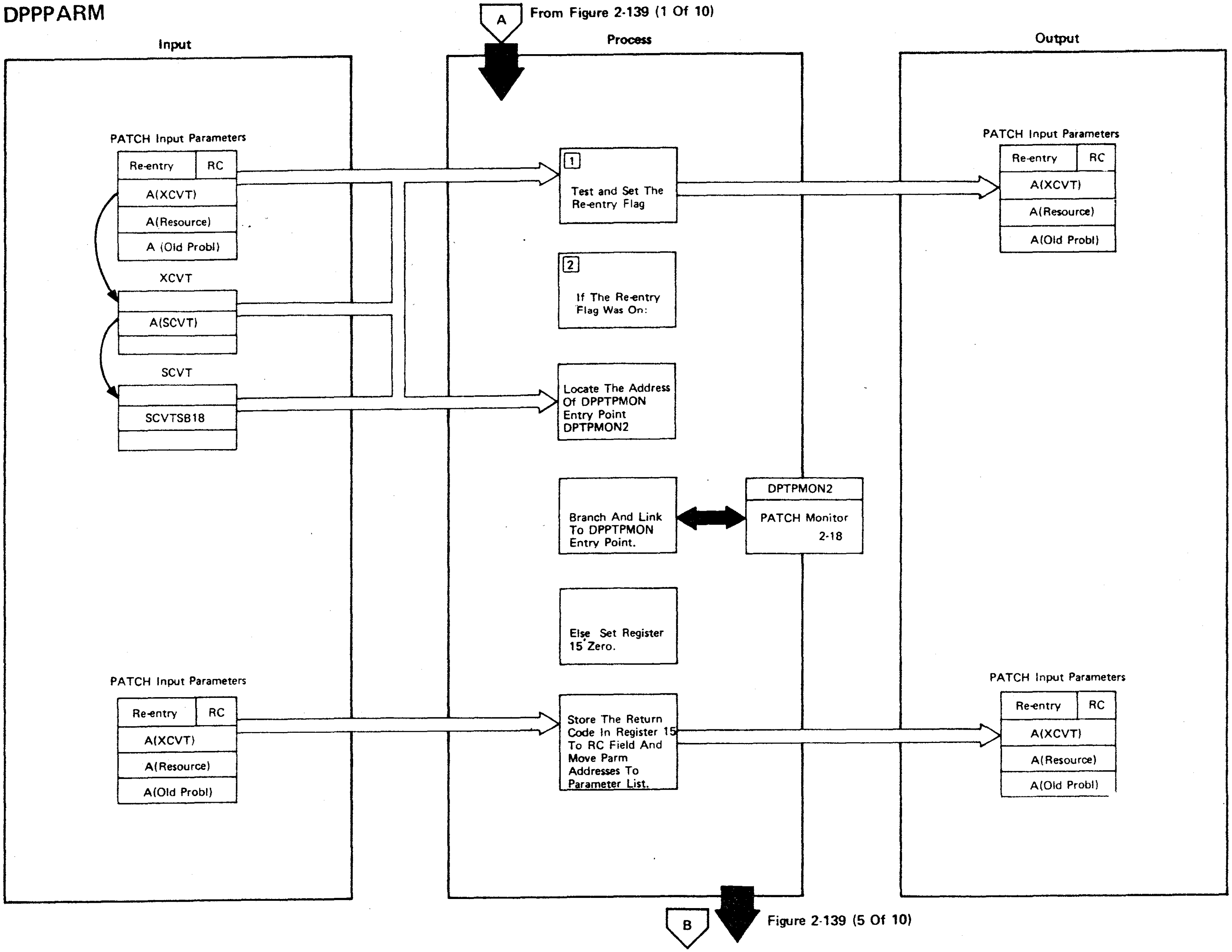


Figure 2-139 (3 Of 10) - High-Level Language PATCH Macro Interface Parameter Build Routine

Figure 2-139 (4 of 10).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Test and set the reentry flag.		DPPPARM
2	If the reentry flag was on, locate the address of DPPTPMON entry point DPTPMON2 from SCVT at location SCVTSB18.		DPPPARM

DPPPARM

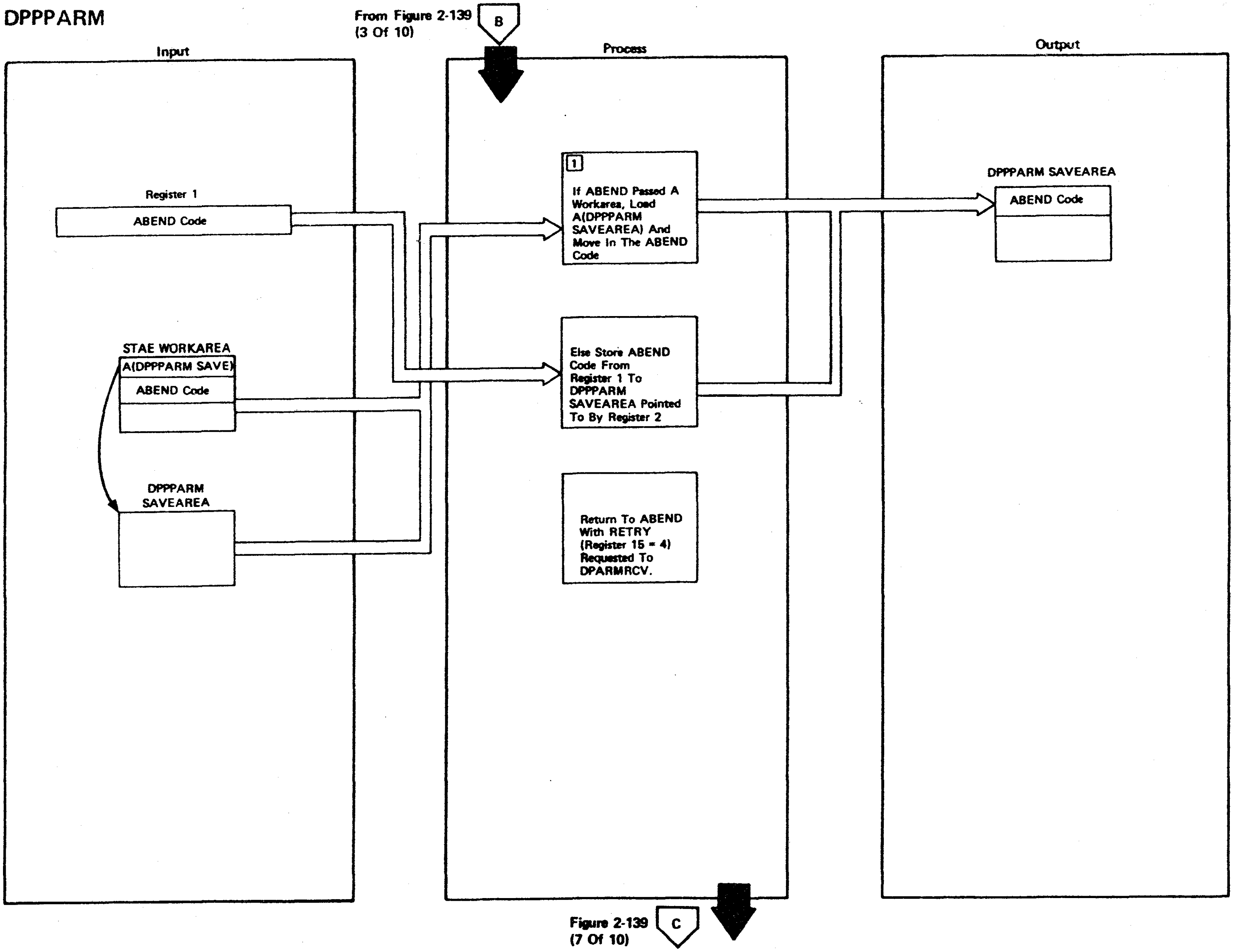

From Figure 2-139
(3 Of 10)
B
Input
Process
Output
Register 1
ABEND Code
STAE WORKAREA
A(DPPPARM SAVE)
ABEND Code
DPPPARM
SAVEAREA
1
If ABEND Passed A Workarea, Load A(DPPPARM SAVEAREA) And Move In The ABEND Code
Else Store ABEND Code From Register 1 To DPPPARM SAVEAREA Pointed To By Register 2
Return To ABEND With RETRY (Register 15 = 4) Requested To DPARMRCV.
DPPPARM SAVEAREA
ABEND Code
Figure 2-139
(7 Of 10)
C

Figure 2-139 (5 Of 10) - High-Level Language PATCH Macro Interface Parameter Build Routine

Figure 2-139 (6 of 10).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If ABEND passed a work area, load address of DPPPARAM save area and move in the ABEND code from the STAE work area; otherwise store the ABEND code from register 1 to DPPPARM save area pointed to by register 2.		DPPPARM

DPPPARM

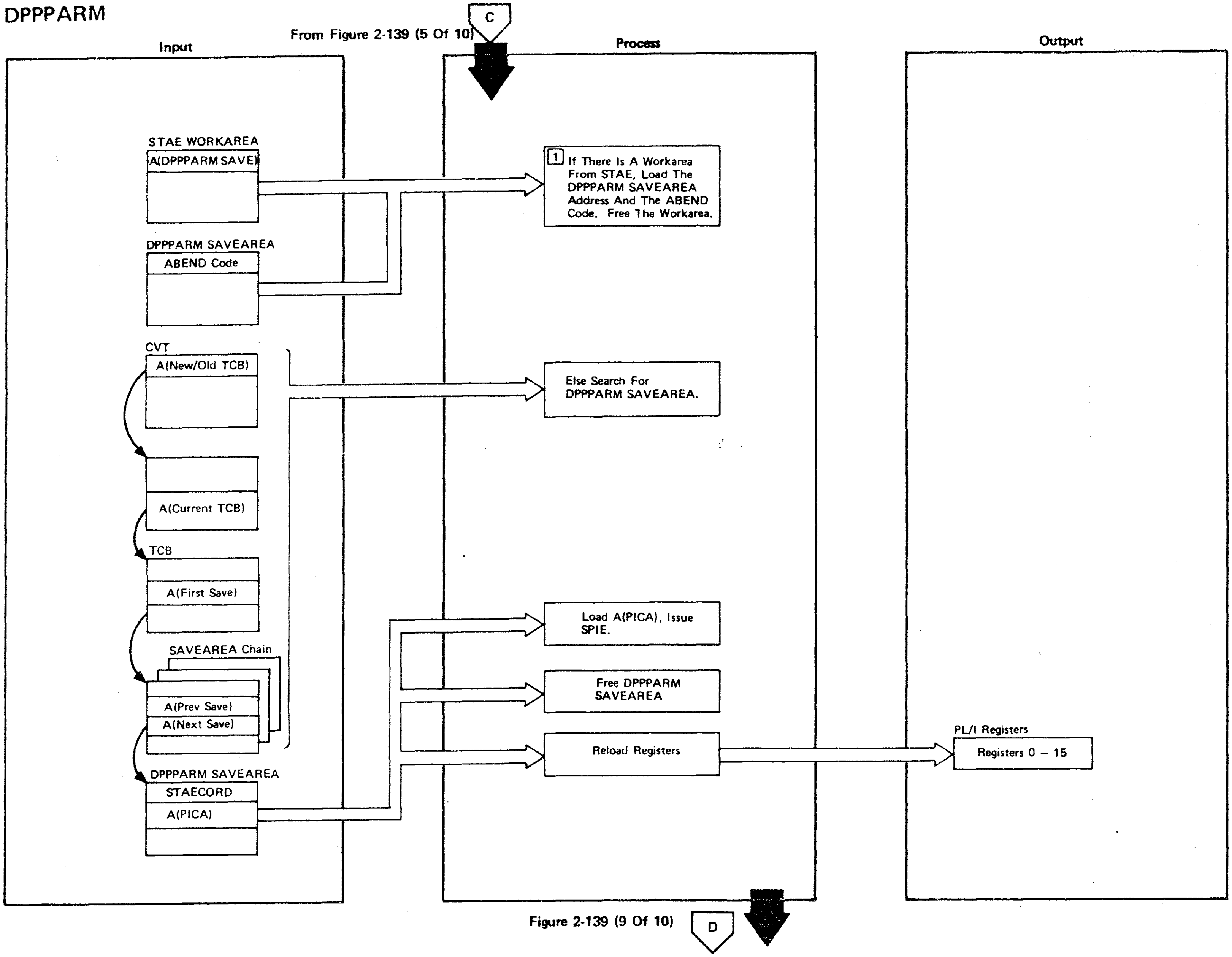


Figure 2-139 (7 Of 10) - High-Level Language PATCH Macro Interface Parameter Build Routine

Figure 2-139 (8 of 10).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If there is a work area from STAE, load the DPPPARM save area address (register 13). Free the work areas; otherwise search for DPPPARM save area.		DPPPARM

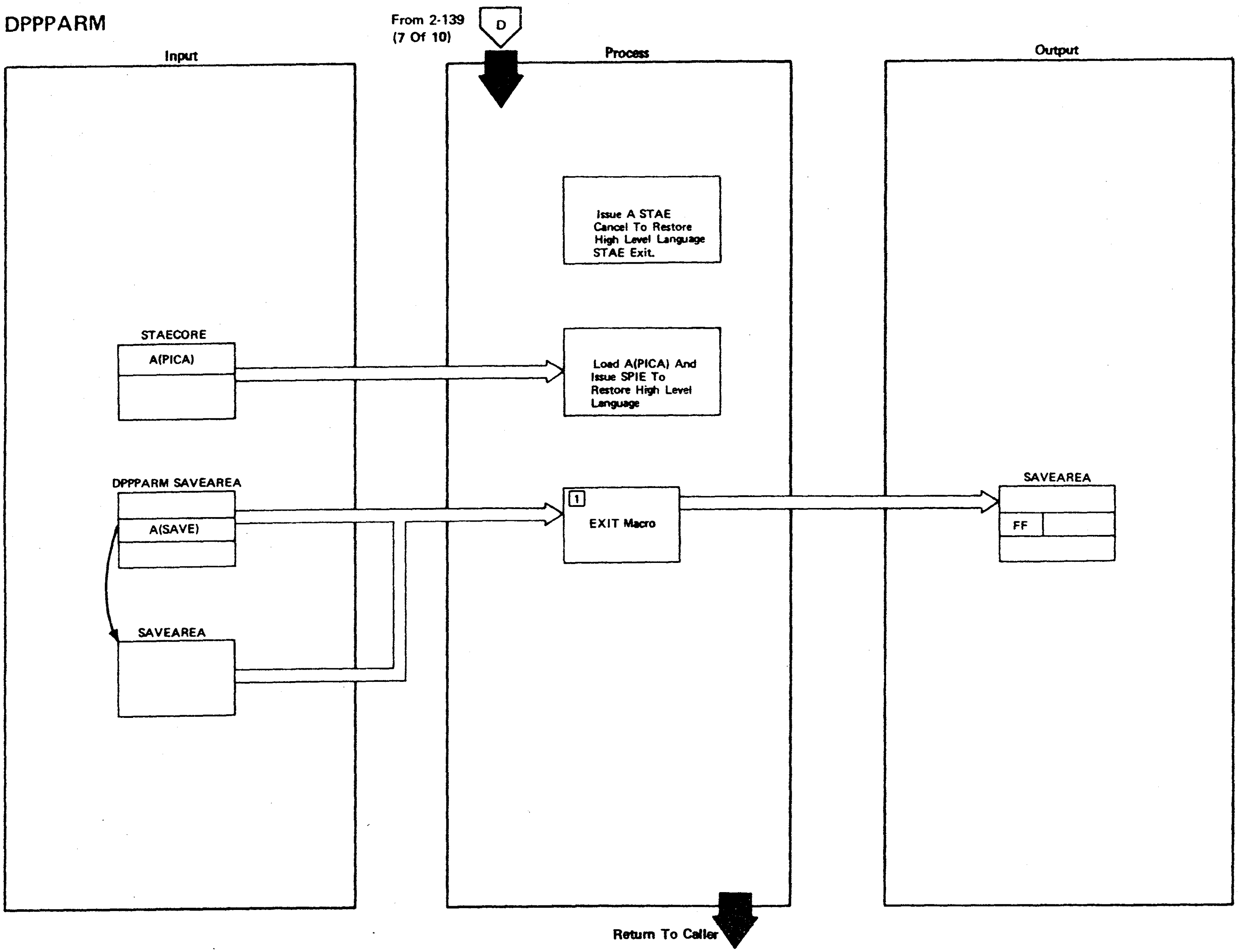


Figure 2-139 (9 Of 10) - High-Level Language PATCH Macro Interface Parameter Build Routine

Figure 2-139 (10 of 10).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The EXIT macro FREEMAINs the save area obtained at entry, marks the previous save area to indicate DPPPARM has completed, reloads the caller's registers, and returns via register 14.		DPPPARM

DPPPLIO

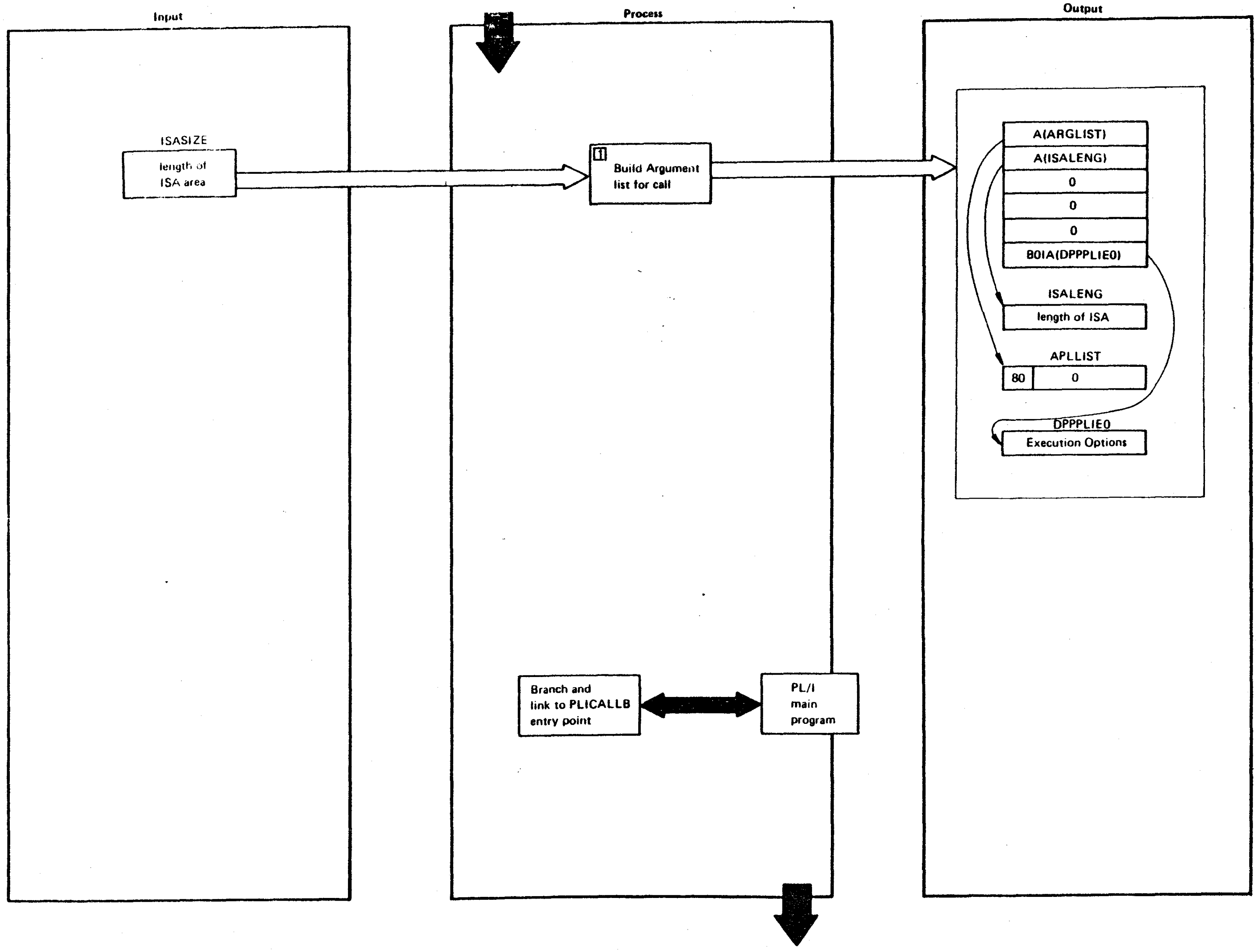


Figure 2-344.1 (1 Of 2) - PL/1 Optimizing Compiler Initialization

Figure 2-139.1 (2 of 2)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If "ISASIZE" external symbol exists and if it is not zero then use the value at ISASIZE, rounded to a double word, and place at ISALENG. If "ISASIZE" external symbol does not exist then use default size.		DPPPLIO

Two CPU Operations

The Special Real Time Operating System has facilities to allow for execution in a two-CPU configuration where a job in the backup CPU monitors the performance of the online CPU. When either CPU recognizes that a failure has occurred, that CPU can request a failover, and the backup CPU becomes the online CPU. Failover can also be initiated by program request to facilitate scheduled maintenance or changes to the operational environment.

Failover/restart operates by copying the contents of virtual storage, the OS/VS1 job queue, and the SWADS for the one or two partitions that encompass the realtime job, into a disk data set. This is initiated by a WTFAILDS macro call from DPPINIT1. The write failover data set SVC routine, DOMIRFLV, is responsible for ensuring that both partitions in a two-partition environment have issued the WTFAILDS macro. Then DOMIRFLV writes the failover data set.

This data set can then be IPLed to accomplish the restart.

The effect of IPLing this volume is to return the System/370 to the identical state it was when the RESTART WRITE card was encountered in the SYSINIT Special Real Time Operating System initialization stream.

The failover/restart bootstrap program, DOMIRBT, is responsible for restoring the virtual storage, the job queue data set, and one or two SWADS data sets to the identical state they were in when the restart was written.

The continuous monitor feature of the Special Real Time Operating System is available in all systems having the failover/restart feature.

The continuous monitor is started by patching a task with EP=DOMIRCMN. This can be done by a user program, by a PATCH card in the SYSINIT input stream, or by the CMON parameter on the RESTART card.

The probe functions, DOMIRPRB, operates in the backup CPU and tests the online CPU (the continuous monitor) and is responsible for recommending failover when it gets a Continuous Monitor Recommended Failover signal or if the continuous monitor fails to change the bits on the static signal lines at the specified rate.

Upon a system restart, any data base data sets supported by DDS must be closed and opened to account for any changes in the status of primary and backup DDS data sets. This is accomplished by an IRB scheduled under the job step task by DPPIIRB. This IRB executes program DPPDWRST, whichcloses and re-opens the data sets.

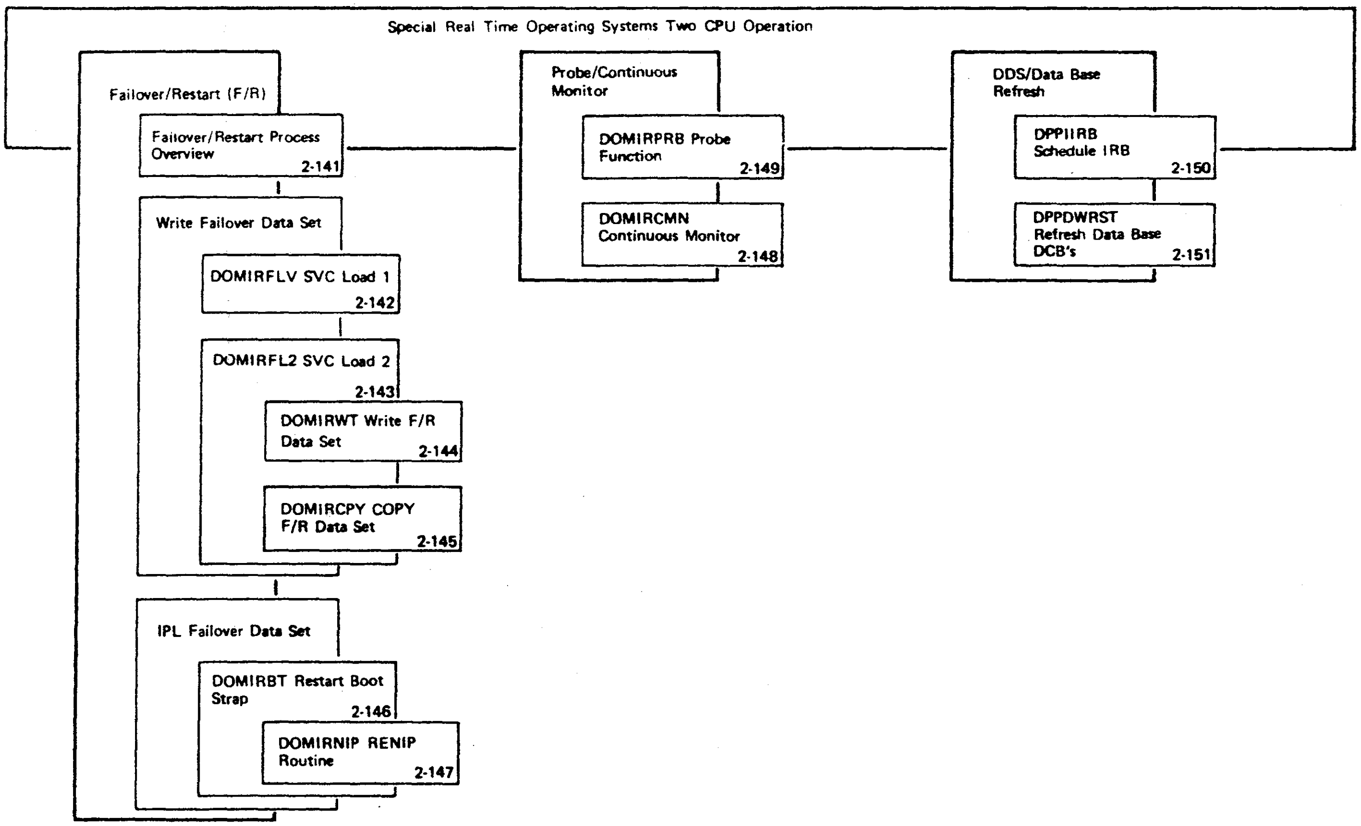

Special Real Time Operating Systems Two CPU Operation
Failover/Restart (F/R)
Failover/Restart Process Overview 2-141
Write Failover Data Set
DOMIRFLV SVC Load 1 2-142
DOMIRFL2 SVC Load 2 2-143
DOMIRWT Write F/R Data Set 2-144
DOMIRCPY COPY F/R Data Set 2-145
IPL Failover Data Set
DOMIRBT Restart Boot Strap 2-146
DOMIRNIP RENIP Routine 2-147
Probe/Continuous Monitor
DOMIRPRB Probe Function 2-149
DOMIRCMN Continuous Monitor 2-148
DDS/Data Base Refresh
DPPIIRB Schedule IRB 2-150
DPPDWRST Refresh Data Base DCB's 2-151

Figure 2-140 - Special Real Time Operating System Two CPU Operation Overview

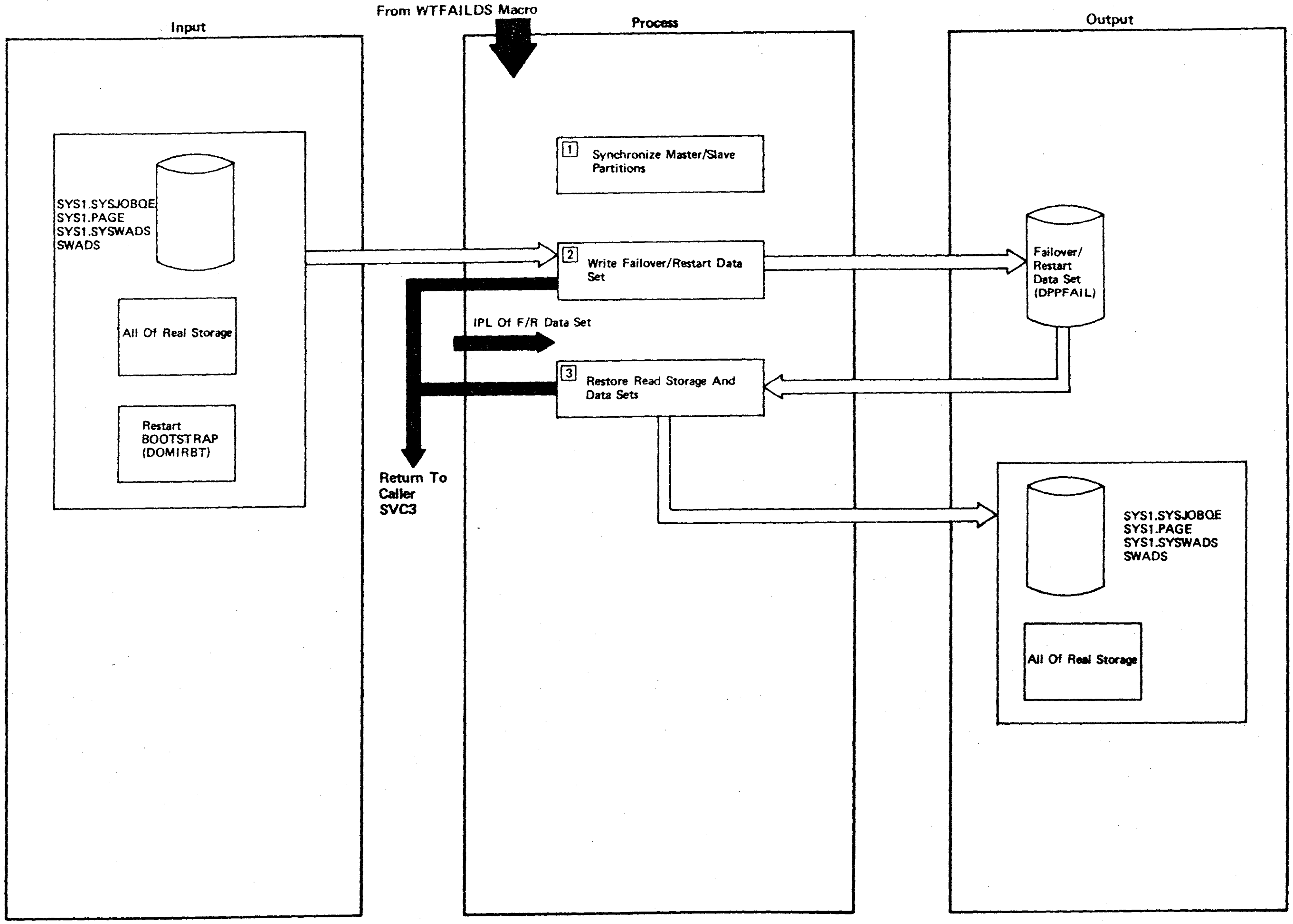


Figure 2-141 - Failover/Restart Process Overview

Figure 2-141 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The MASTER (SLAVE) partition delays until the SLAVE (MASTER) partition issues the WTFAILDS macro. Execution then continues under the MASTER SLAVE partition.		
2	All of real storage, the protect keys, the SYS1.SYSJOBQE data set(s), the SYS1.SYSWADS data set, the active entries in the SYS1.PAGE data set(s), and one or two SWADS (MASTER and SLAVE) data sets are copied to the DPPFAIL data set. The IPL1 and IPL2 records on track zero are modified to read in module DOMIRBT when the disk is IPLed.		
3	When the disk containing the data set named in the DPPFAIL DD card is IPLed, DOMIRBT receives control and restores real storage, the protect keys, SYS1. SYSJOBQE, SYS1.SYSWADS and the active SYS1.PAGE entries. SYS1.SYSPOOL is not restored. Control is returned to the MASTER and SLAVE partitions just as it was when the original WTFAILDS was issued except that return code 4 is placed in register 15 and the restart flags in the XCVT are set to reflect an IPL of the failures/ restart data set on the same or a different CPU.		

DOMIRFLV

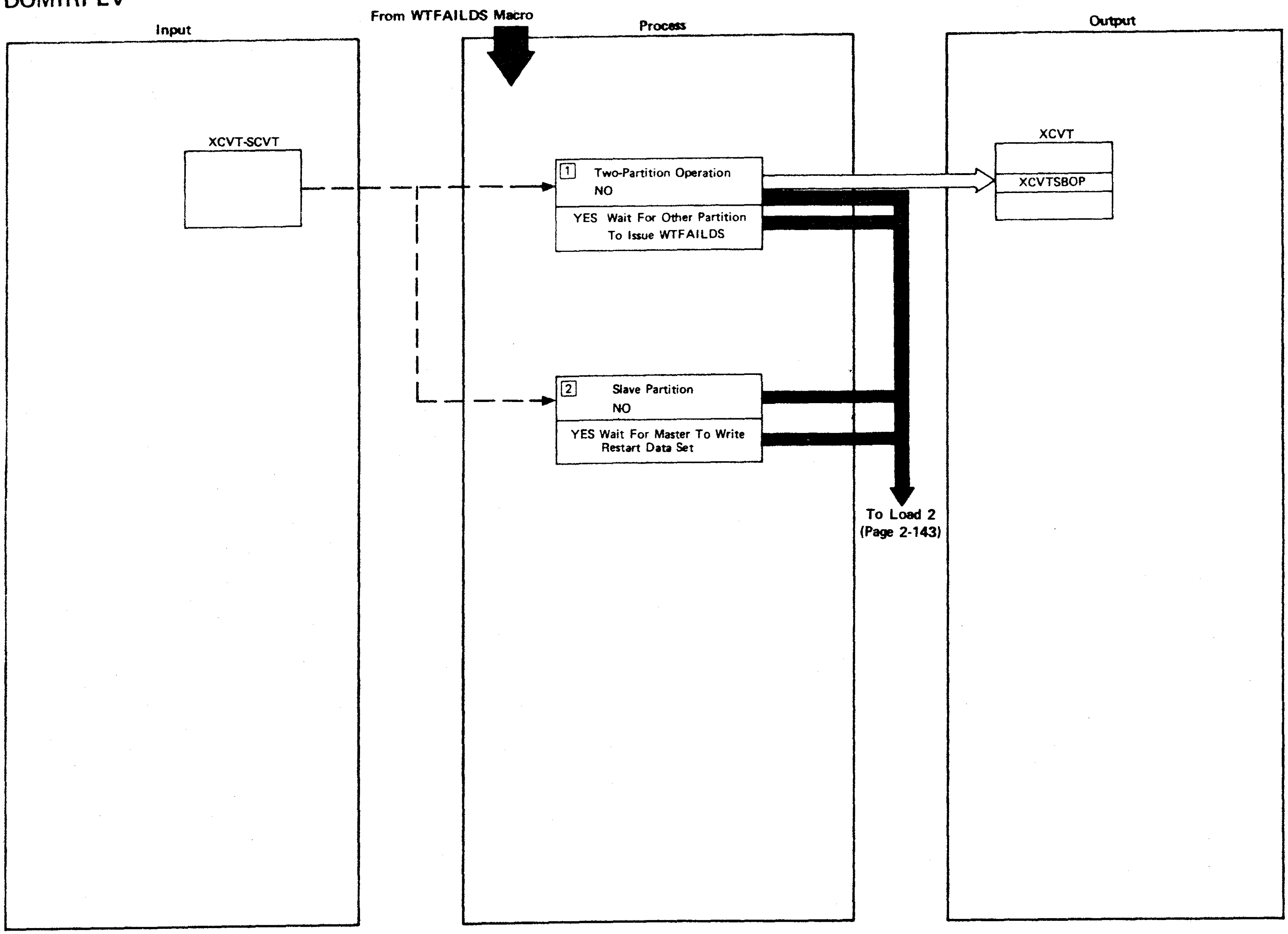

Input
From WTFAILDS Macro
Process
Output
XCVT-SCVT
1 Two-Partition Operation
NO
YES Wait For Other Partition To Issue WTFAILDS
2 Slave Partition
NO
YES Wait For Master To Write Restart Data Set
To Load 2 (Page 2-143)
XCVT
XCVTSBOP

Figure 2-142 (1 Of 2) - Failover/Restart SVC - Load 1

Figure 2-142 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The SCVTFLG1 field is tested for two-partition operation. ABEND 37 is issued if the other partition cannot be located. ABEND 38 is issued if another WTFAILDS is still in progress. ABEND 39 is issued if a realtime initialization has not been performed. An ENQ is issued on QNAME 'DPPINIT', and RNAME of the MASTER realtime jobs to determine if the other partition has reached its WTFAILDS yet. If it has, its ECB (in the SVRB) is posted; otherwise, this partition waits on an ECB in the SVRB that the other partition will post.	USER 37 USER 38 USER 39	DOMIRFLV
2	The SLAVE partition waits on an ECB in the SVRB which the MASTER partition will post when the restart is completed.		DOMIRFLV

DOMIRFL2

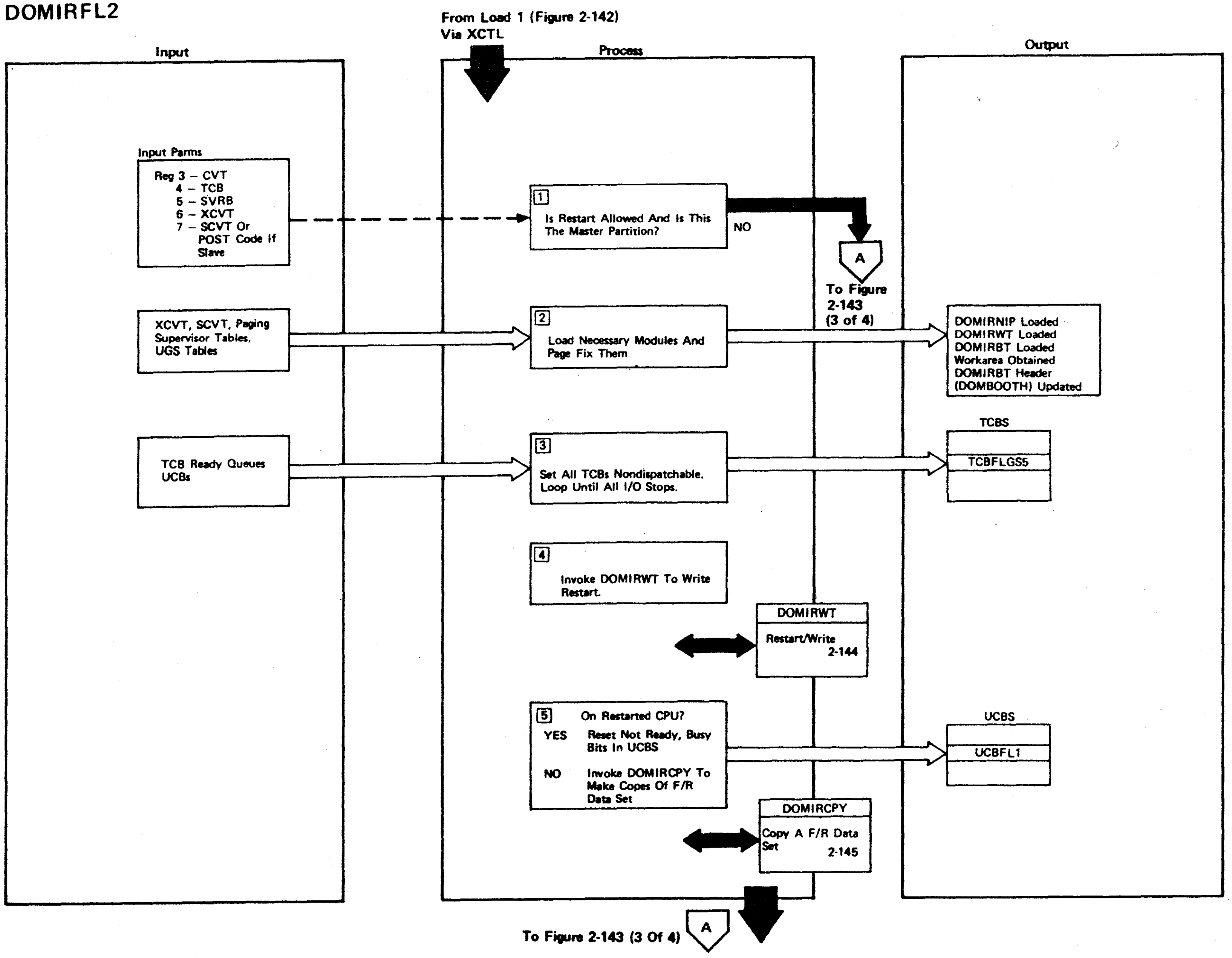

From Load 1 (Figure 2-142)
Via XCTL
Input
Process
Output
Input Parms
Reg 3 – CVT
4 – TCB
5 – SVRB
6 – XCVT
7 – SCVT Or POST Code If Slave
1
Is Restart Allowed And Is This The Master Partition?
NO
A
To Figure 2-143 (3 of 4)
XCVT, SCVT, Paging Supervisor Tables, UGS Tables
2
Load Necessary Modules And Page Fix Them
DOMIRNIP Loaded
DOMIRWT Loaded
DOMIRBT Loaded
Workarea Obtained
DOMIRBT Header (DOMBOOTH) Updated
TCB Ready Queues
UCBs
3
Set All TCBs Nondispatchable. Loop Until All I/O Stops.
TCBS
TCBFLGS5
4
Invoke DOMIRWT To Write Restart.
DOMIRWT
Restart/Write
2-144
5
On Restarted CPU?
YES
Reset Not Ready, Busy Bits In UCBS
NO
Invoke DOMIRCPY To Make Copes Of F/R Data Set
UCBS
UCBFL1
DOMIRCPY
Copy A F/R Data Set
2-145
To Figure 2-143 (3 Of 4)
A

Figure 2-143 (1 Of 4) - Failover/Restart SVC - Load 2

Figure 2-143 (2 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If this is the SLAVE partition or if this job does not own restart/write eligibility (CINFD table), control is passed to step 1, Figure 2-143 (3 of 4).		DOMIRFL2
2	A work area the size of the maximum blockage of the device containing the failures/restart data set is obtained. Modules DOMIRWT (restart/write), DOMIRBT (restart bootstrap), and DOMIRNIP (re-NIP) are loaded and page-fixed. The DOMIRBT header DOMBOOTH is initialized.		DOMIRFL2
3	All TCBs except the one under which DOMIRFL2 is executing are set nondispatchable using the TCBSYS bit. A loop is entered to test for all I/O complete by testing the UCBPST and UCBBSY bits of all UCBs.		DOMIRFL2
4	DOMIRWT is invoked via BALR to write the failures/restart data set.		DOMIRFL2
5	The return code from DOMIRWT is tested. Code 4 indicates that an IPL of the failures/restarts data set has occurred. If this is not a restarted CPU, DOMIRCPY is loaded and BALRed to make copies (if any) of the failures/restart data set.		DOMIRFL2

DOMIRFL2

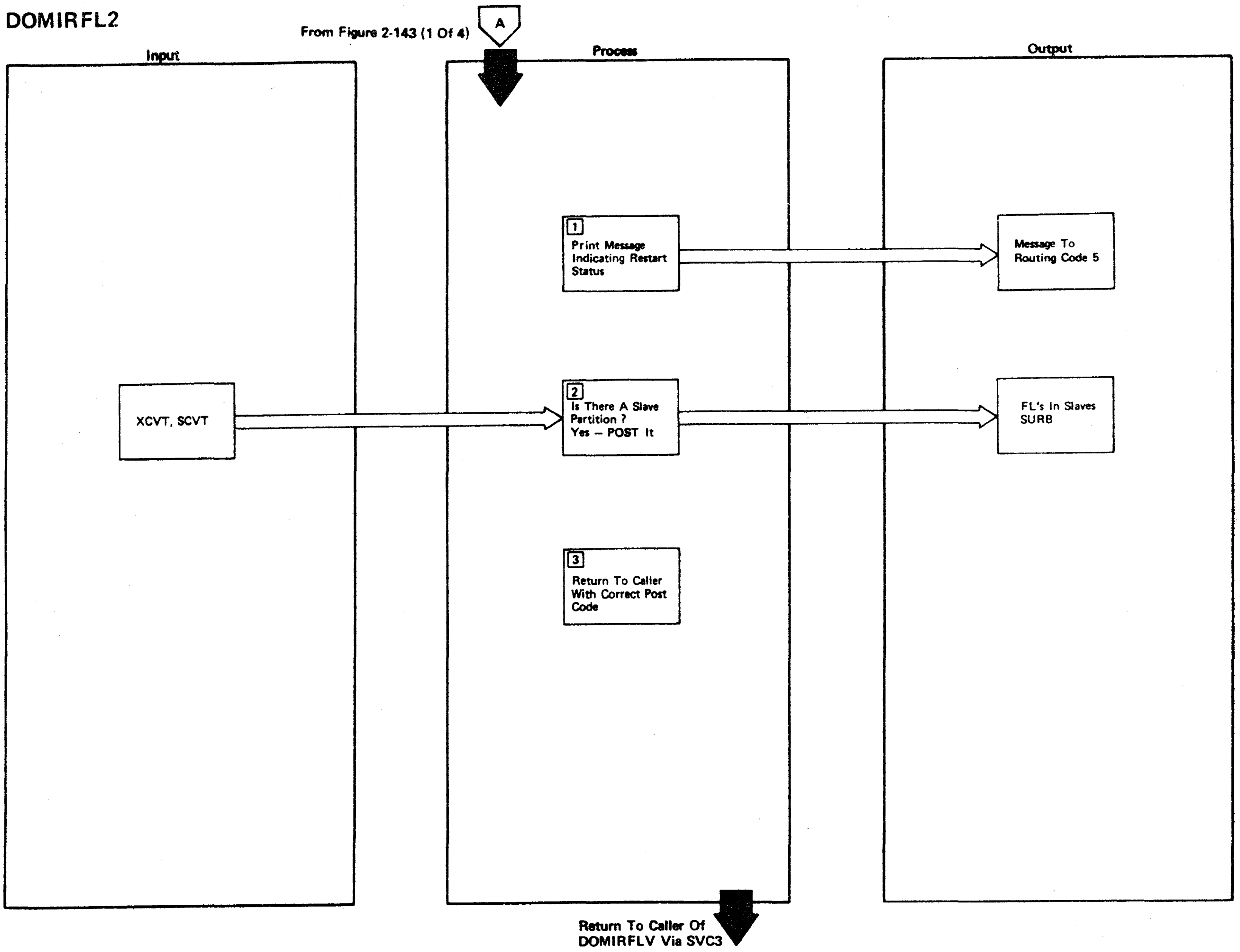

From Figure 2-143 (1 Of 4)
A
Input
Process
Output
1
Print Message Indicating Restart Status
Message To Routing Code 5
XCVT, SCVT
2
Is There A Slave Partition ? Yes – POST It
FL's In Slaves SURB
3
Return To Caller With Correct Post Code
Return To Caller Of DOMIRFLV Via SVC3

Figure 2-143 (3 Of 4) - Failover/Restart SVC - Load 2

Figure 2-143 (4 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	A message to indicate the success or failure of the failures/restart write is printed.	DPP080A DPP081A DPP082A DPP083A DPP084A DPP085A DPP086A DPP087A DPP090I DPP091I DPP092A DPP093I	DOMIRFL2
2	If a SLAVE partition is waiting, its ECB is posted.		DOMIRFL2
3	The caller of DOMIRFLV is returned to via EXIT (SVC3).		DOMIRFL2

DOMIRWT

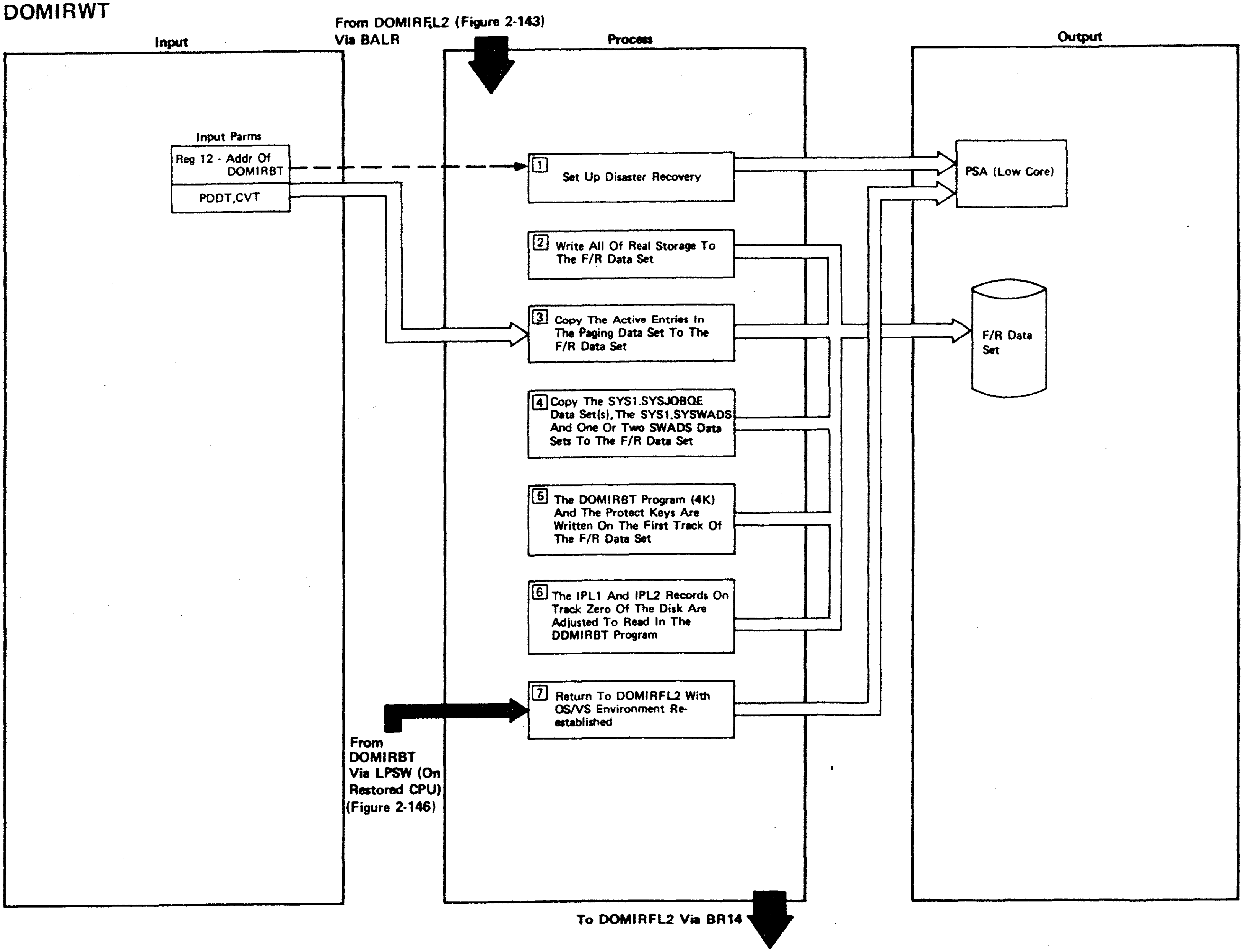

From DOMIRFL2 (Figure 2-143)
Via BALR
Input
Process
Output
Input Parms
Reg 12 - Addr Of DOMIRBT
PDDT,CVT
1 Set Up Disaster Recovery
2 Write All Of Real Storage To The F/R Data Set
3 Copy The Active Entries In The Paging Data Set To The F/R Data Set
4 Copy The SYS1.SYSJOBQE Data Set(s), The SYS1.SYSWADS And One Or Two SWADS Data Sets To The F/R Data Set
5 The DOMIRBT Program (4K) And The Protect Keys Are Written On The First Track Of The F/R Data Set
6 The IPL1 And IPL2 Records On Track Zero Of The Disk Are Adjusted To Read In The DDMIRBT Program
7 Return To DOMIRFL2 With OS/VS Environment Re-established
From DOMIRBT Via LPSW (On Restored CPU) (Figure 2-146)
PSA (Low Core)
F/R Data Set
To DOMIRFL2 Via BR14

Figure 2-144 (1 Of 4) - Failover/Restart WRITE

Figure 2-144 (2 Of 4)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	As a non-OS/VS1 environment exists (standalone I/O) during DOMIRWT's execution, certain precautions are taken to ensure that it can finish execution. The program check new PSW and machine check new PSW are saved and replaced with PSWs that point to recovery routines within DOMIRWT. The DAT box remains on throughout the execution of DOMIRWT. The control registers are saved in the DOMIRBT header.		DOMIRWT
2	All of real storage from location 0 to the end (as determined from the CVT) excepting the 4K occupied by DOMIRBT and the first 2K of the work area are written to the failures/restart data set starting with the second track. The protect key of each block is saved and restored across each I/O operation to avoid changing the reference bit.		DOMIRWT
3.	The active entries on the paging data set(s) (those which could possibly be paged in) as preserved on the failures/restart data set. An LRA instruction is issued for all address space (2K at a time) defined in the PDDT. A paging data set entry is copied to the failures/restart data set if either of the following conditions occurs on the LRA: a. The LRA indicates a page exception and the "valid bit" (bit 15) is set in the page table entry. b. The LRA indicates a valid translation and the change bit is off in the frame.		DOMIRWT

Figure 2-144 (3 Of 4)

Step	Extended Description	Messages and ABEND Codes	PDL Segment
4	The entire contents of the JOBQUEUE and SYSWADS are copied to the failover/restart data set. Unless the partition is using SWA, the SWADS data set(s) are also copied.		DOMIRWT
5	The two 2K blocks of DOMIRBT are written as one record on track zero of the failover/restart data set. The protect keys for all of real storage are saved and written as the second record on track zero of the data set.		DOMIRWT
6	The IPL1 and IPL2 records are adjusted to read DOMIRBT into the same real storage locations it was written from during the disk IPL. The PSW placed in the IPL1 record points to the first byte after the DOMIRBT header (real address).		DOMIRBT
7	The PSA is restored and control is returned to DOMIRFL2. A code of zero is passed in register 15 if the restart/write was successful. A code of 12 is passed if an error occurred with a subcode in register zero. If this is a restarted CPU, DOMIRBT will have entered DOMIRWT at location RESUME after restoring the environment as it was at restart/write. In this case, code 4 is returned to DOMIRFL2.		DOMIRBT

Intentionally Blank

Figure 2-144 (4 of 4)

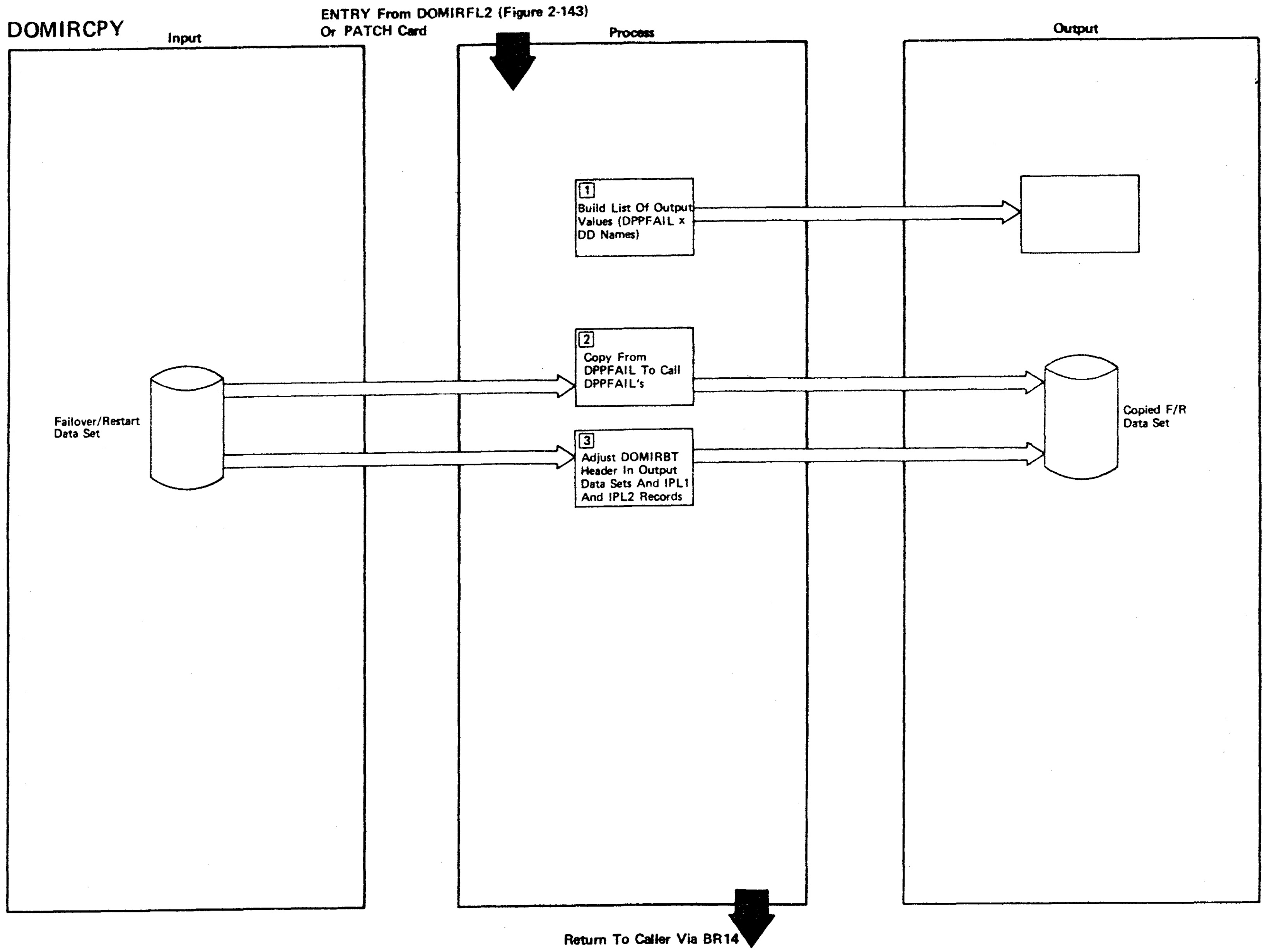

DOMIRCPY
Input
ENTRY From DOMIRFL2 (Figure 2-143)
Or PATCH Card
Process
Output
1
Build List Of Output Values (DPPFAIL x DD Names)
2
Copy From DPPFAIL To Call DPPFAIL's
3
Adjust DOMIRBT Header In Output Data Sets And IPL1 And IPL2 Records
Failover/Restart Data Set
Copied F/R Data Set
Return To Caller Via BR14

Figure 2-145 (1 Of 2) - Failover/Restart Data Set Copy

Figure 2-14 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	A list of the output volumes (DPPFAILx) is constructed. All must be of the same device type and only one failover/restart data set can exist on a volume. In addition, a failover/restart data set cannot reside on the system residence volume that contains SYS1.NUCLEUS.	DPP089A	DOMIRCPY
2	The data set is copied, track for track, using EXCP from the DPPFAIL volume to each DPPFAILx volume.	DPP088A	DOMIRCPY
3	The DOMIRBT header is adjusted on each output volume for new absolute disk addresses. The IPL1 and IPL2 records are also adjusted.		DOMIRCPY

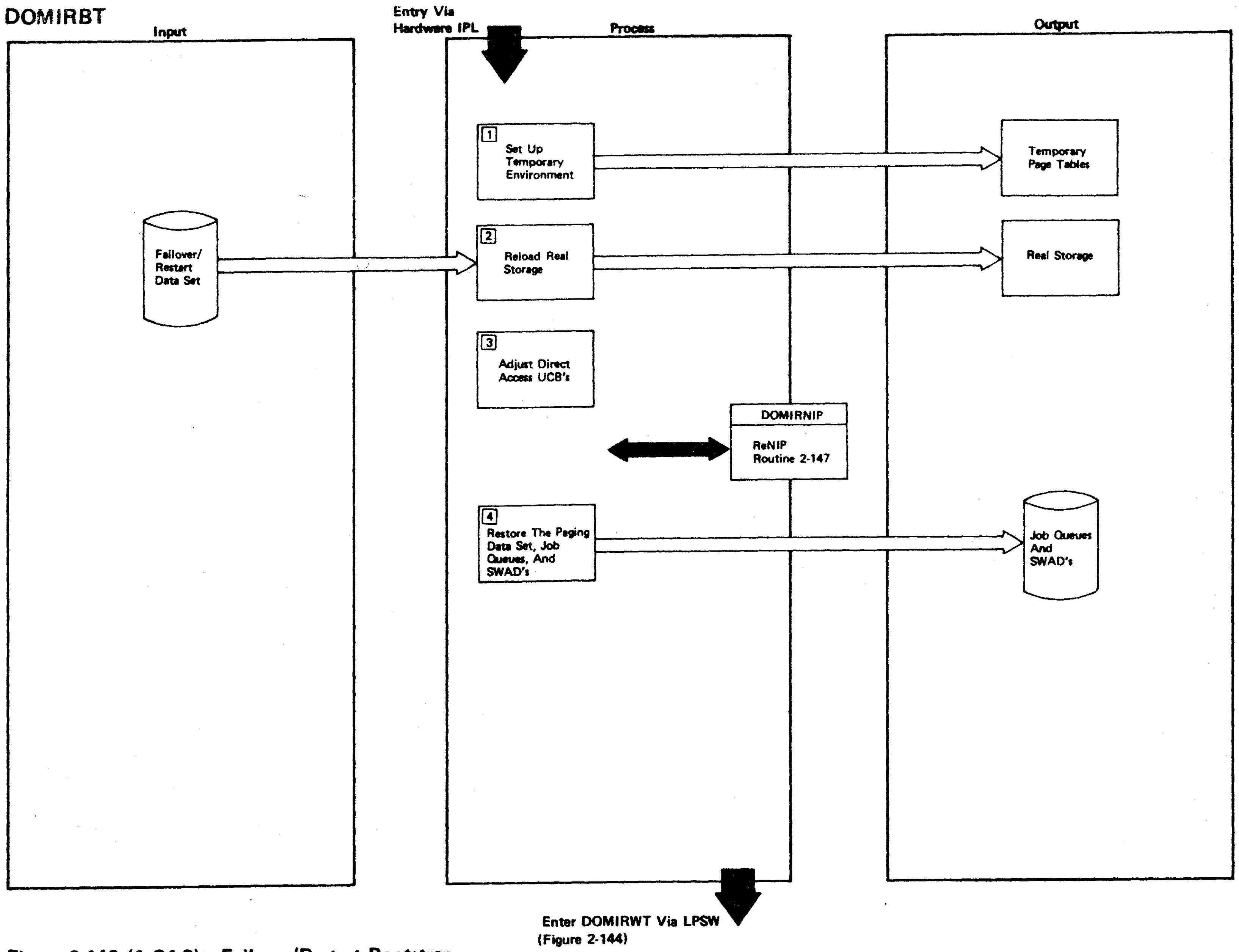
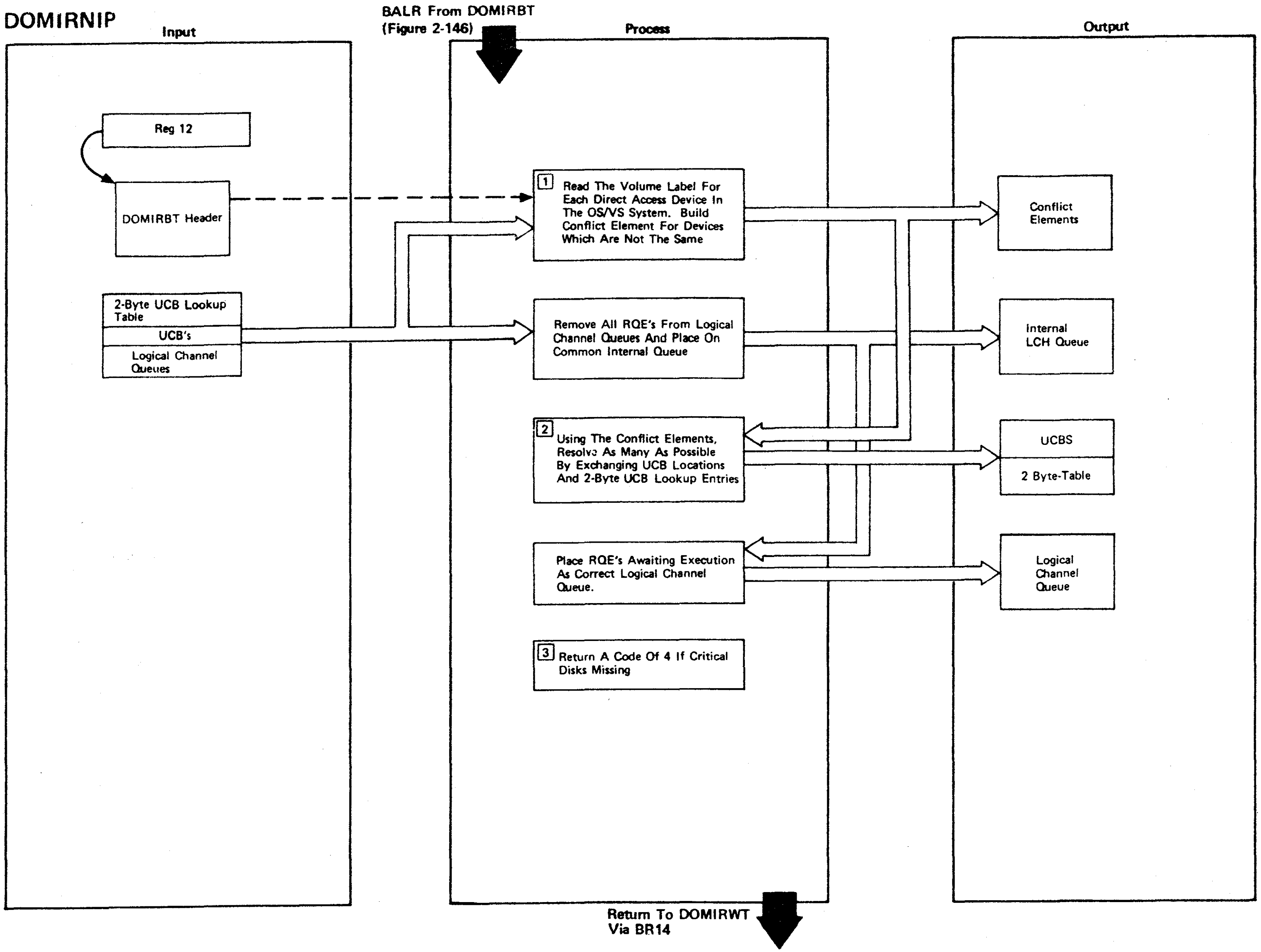


Figure 2-146 (1 Of 2) - Failover/Restart Bootstrap

Figure 2-146 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	A check is made to determine if this CPU has enough real storage. If not, a X'18' wait occurs. A set of page tables is constructed with 8K of valid address space: The first 2K of the PSA, the 4K of the DOMIRBT module, and the first 2K of the work area. Control register 0 and 1 are initialized, and the DAT box is turned on. A machine check in the restart bootstrap causes a X'E2' wait state. An unexpected program check causes a X'18' wait state or a X'19' wait state. The location of the Format #1 DSCB for the failover/restart data set is read in and if the data set no longer exists, a X'F' wait state occurs.	"Failover/ Restart in Progress" on a 3210 or 3215.	DOMIRBT
2	All of real storage is reloaded from the failover/restart data set. After real storage has been reloaded, the control registers are restored to their values at the time the failover/restart data set was written.		DOMIRBT
3	DOMIRNIP is invoked via a BALR to check for disks mounted on addresses different than they were at time the failover/restart data set was written. If DOMIRNIP indicates that a needed disk is missing, a X'd' wait state occurs.		DOMIRBT
4	The paging data set(s), SYS1.SYSJOBQE, and SYS1.SYSWADS are restored. The SWADS data sets (MASTER and SLAVE) are restored unless SWA was used.		DOMIRBT

DOMIRNIP



1 Read The Volume Label For Each Direct Access Device In The OS/VS System. Build Conflict Element For Devices Which Are Not The Same

Remove All RQE's From Logical Channel Queues And Place On Common Internal Queue

2 Using The Conflict Elements, Resolvə As Many As Possible By Exchanging UCB Locations And 2-Byte UCB Lookup Entries

Place RQE's Awaiting Execution As Correct Logical Channel Queue.

3 Return A Code Of 4 If Critical Disks Missing

Figure 2-147 (1 Of 2) - ReNIP Routine

Figure 2-147 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Conflict elements built for volumes which have moved to a different device address are built into the failover/restart work area.		DOMIRNIP
2	Conflicts are resolved by swapping the location of the UCBs and their pointers in the 2-byte lookup table. Thus, DEBs and other UCB pointers throughout OS/VS1 still point to the same volume and IOS can still find the correct UCB based on device address. Device related fields, e.g., UCBLCI, go with the UCB to the new location. Volume related fields, e.g., UCBDMCT, remain in the old location.		DOMIRNIP
3	A critical disk is one that is permanently resident and allocated.		DOMIRNIP

DOMIRCMN

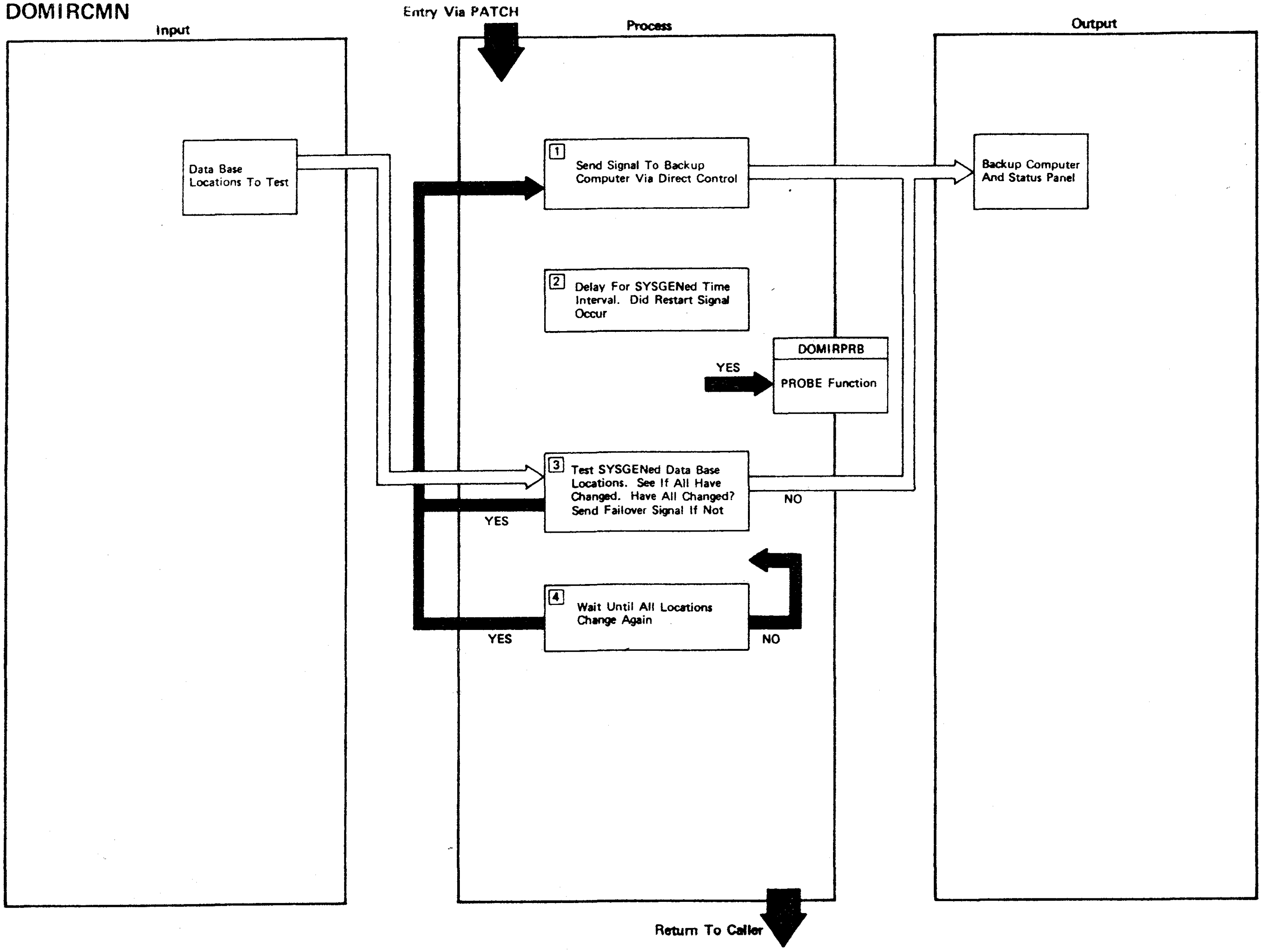

Entry Via PATCH
Input
Process
Output
Data Base Locations To Test
1 Send Signal To Backup Computer Via Direct Control
Backup Computer And Status Panel
2 Delay For SYSGENed Time Interval. Did Restart Signal Occur
DOMIRPRB
YES
PROBE Function
3 Test SYSGENed Data Base Locations. See If All Have Changed. Have All Changed? Send Failover Signal If Not
YES
NO
4 Wait Until All Locations Change Again
YES
NO
Return To Caller

Figure 2-148 (1 Of 2)- Continuous Monitor

Figure 2-148 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the continuous monitor is operating in duplex mode (no other continuous monitor or probe), a signal is sent to the backup computer via the direct control static data lines. The computer status panel is also driven (ready light).	DPP099I	DOMIRCMN
2	Delay is via STIMER. If a failover request external interrupt occurs, it is interpreted as a restart signal, and an XCTL is executed to DOMIRIPL (within DOMIRPRB) to cause the failover/restart data set to be IPLed.		DOMIRCMN
3	If any of the participating data base locations have failed to change during two time intervals, failover is recommended to the backup computer by sending X'F' on the static data lines.	DPP098A	DOMIRCMN
4	If the data base locations start to change again, the sampling process is restarted.		DOMIRCMN

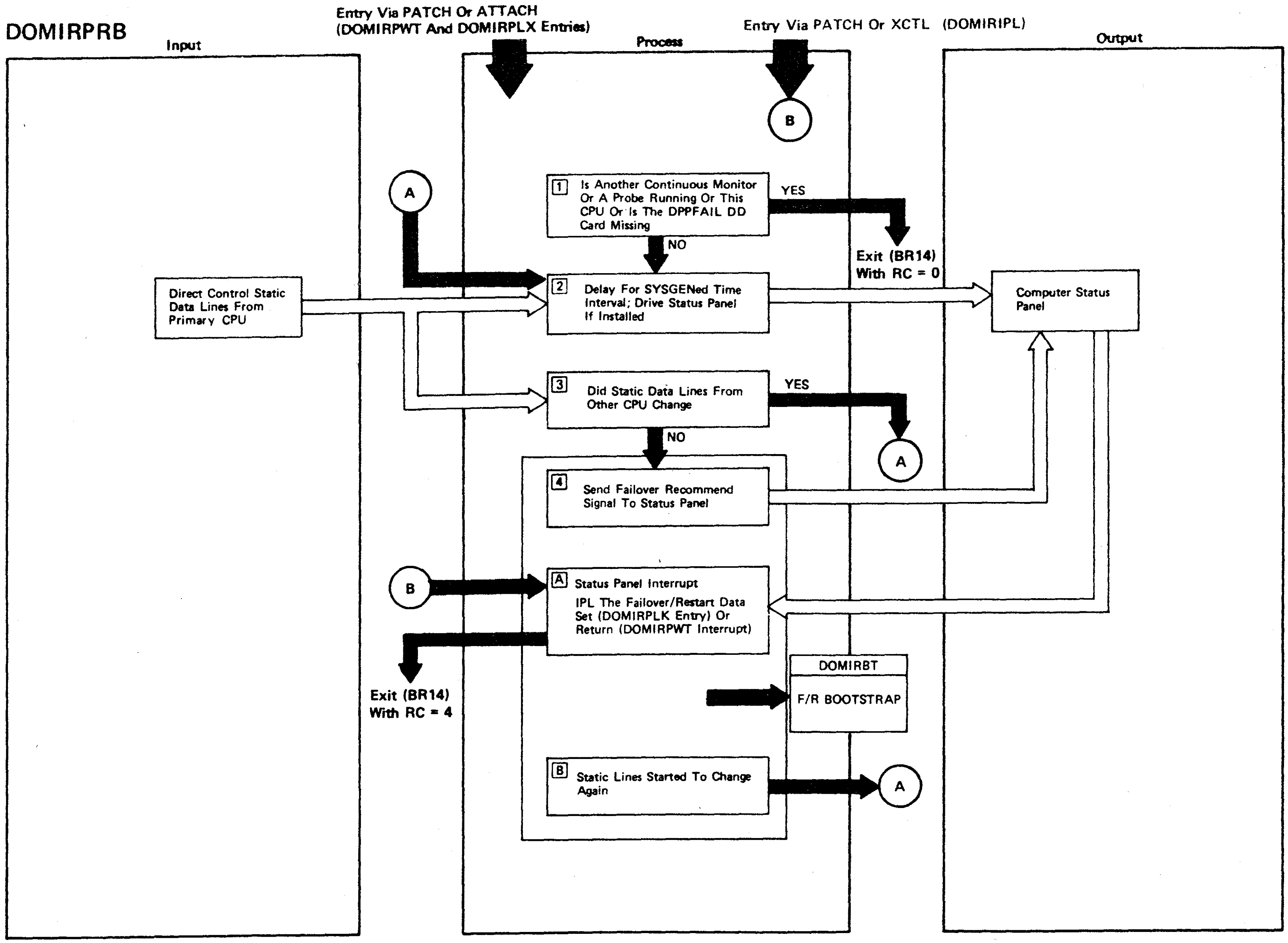

DOMIRPRB
Input
Entry Via PATCH Or ATTACH (DOMIRPWT And DOMIRPLX Entries)
Process
Entry Via PATCH Or XCTL (DOMIRIPL)
Output
B
A
1 Is Another Continuous Monitor Or A Probe Running Or This CPU Or Is The DPPFAIL DD Card Missing
YES
NO
Exit (BR14) With RC = 0
Direct Control Static Data Lines From Primary CPU
2 Delay For SYSGENed Time Interval; Drive Status Panel If Installed
Computer Status Panel
3 Did Static Data Lines From Other CPU Change
YES
NO
A
4 Send Failover Recommend Signal To Status Panel
B
A Status Panel Interrupt
IPL The Failover/Restart Data Set (DOMIRPLK Entry) Or Return (DOMIRPWT Interrupt)
Exit (BR14) With RC = 4
DOMIRBT
F/R BOOTSTRAP
B Static Lines Started To Change Again
A

Figure 2-149 (1 Of 2) - Probe Function

Figure 2-149 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	An ENQ is issued to test for the presence of another continuous monitor or probe. The module terminates if another one is present.		DOMIRPRB
2	The ready light on the status panel is driven, and a delay for the SYSGENed time interval is instigated via the STIMER function.		DOMIRPRB
3	The static lines are tested via READ-DIRECT. If they have changed, the STIMER loop is reissued. If they have failed to change for two STIMER cycles, the failover recommend signal is sent to the computer status panel.		DOMIRPRB
4	The failover confirmed signal from the status panel can occur in response to the failover recommend signal or asynchronously at any time. If the probe was entered at DOMIRPLK, it simulates a hardware IPL to the failover/restart data set. If the probe was entered at DOMIRPWT, it returns with code 4 in register 15. The IPL of the failover/restart data set can also be entered (from DOMIRCMN) at DOMIRIPL. If the static data lines start to change again while awaiting the failover confirmed interrupt, the STIMER loop is reentered.		DOMIRPRB

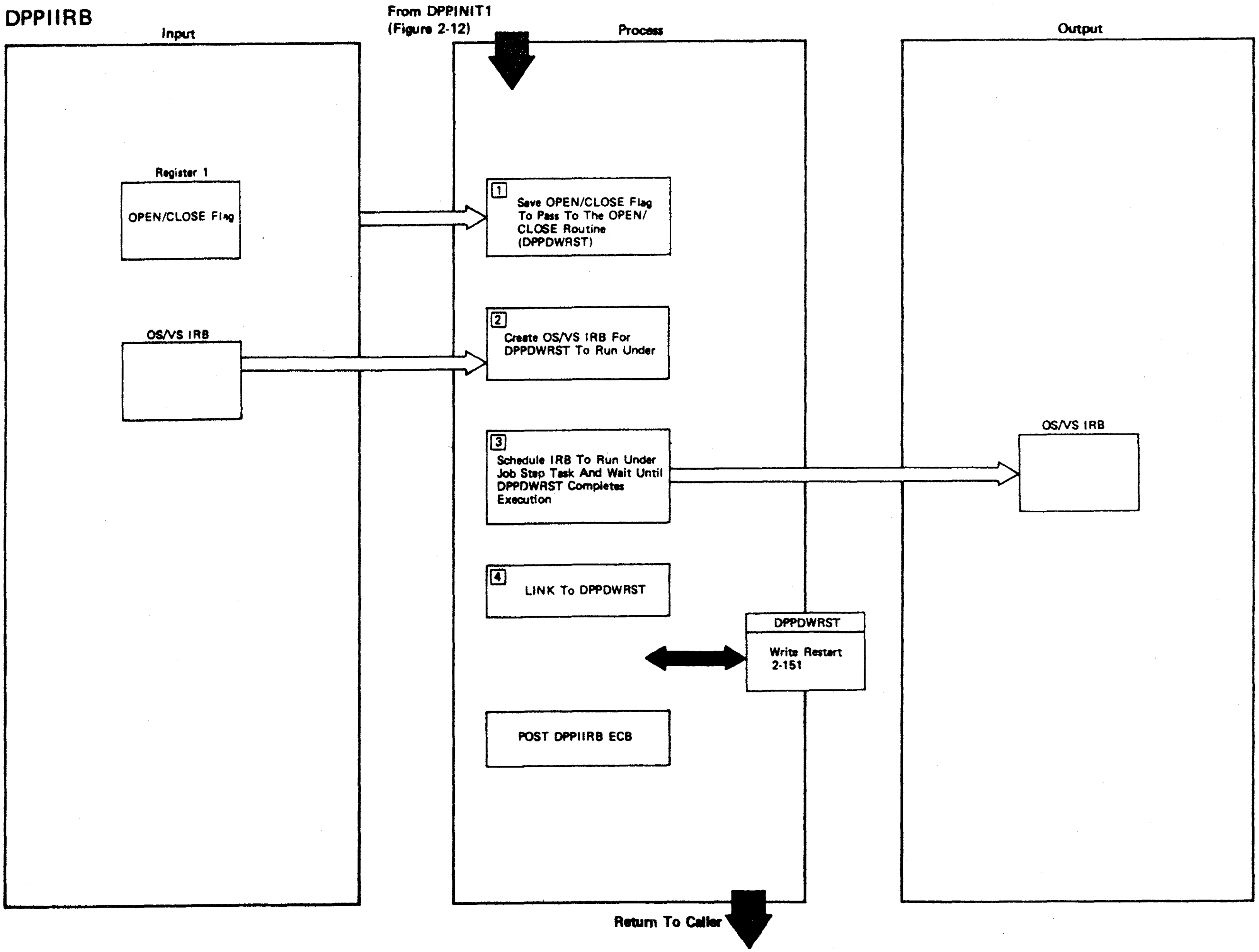


Figure 2-150 (1 Of 2) - Data Base Create IRB Routine

Figure 2-150 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	When entered, register 1 will contain an OPEN/CLOSE flag. If register 1 contains a 0, all data base data sets are closed, and if register 1 contains a one, all data base data sets are opened.		DPPIIRB
2	A CIRB macro will be executed to create an OS/VS1 IRB under which the OPEN/CLOSE routine (DPPDWRST) can run.		DPPIIRB
3	The IRB will be scheduled under the job step task to facilitate the opening or closing of the data base data sets because the initialization task under which DPPIIRB runs, terminates at the end of initialization. DPPIIRB will wait for the OPEN/CLOSE routine to complete.		DPPIIRB
4	When the CIRB macro is executed, it points to a segment in DPPIIRB which links to the OPEN/CLOSE routine. On return from the link, this segment will post the ECB that DPPIIRB is waiting on.		DPPIIRB

DPPDWRST

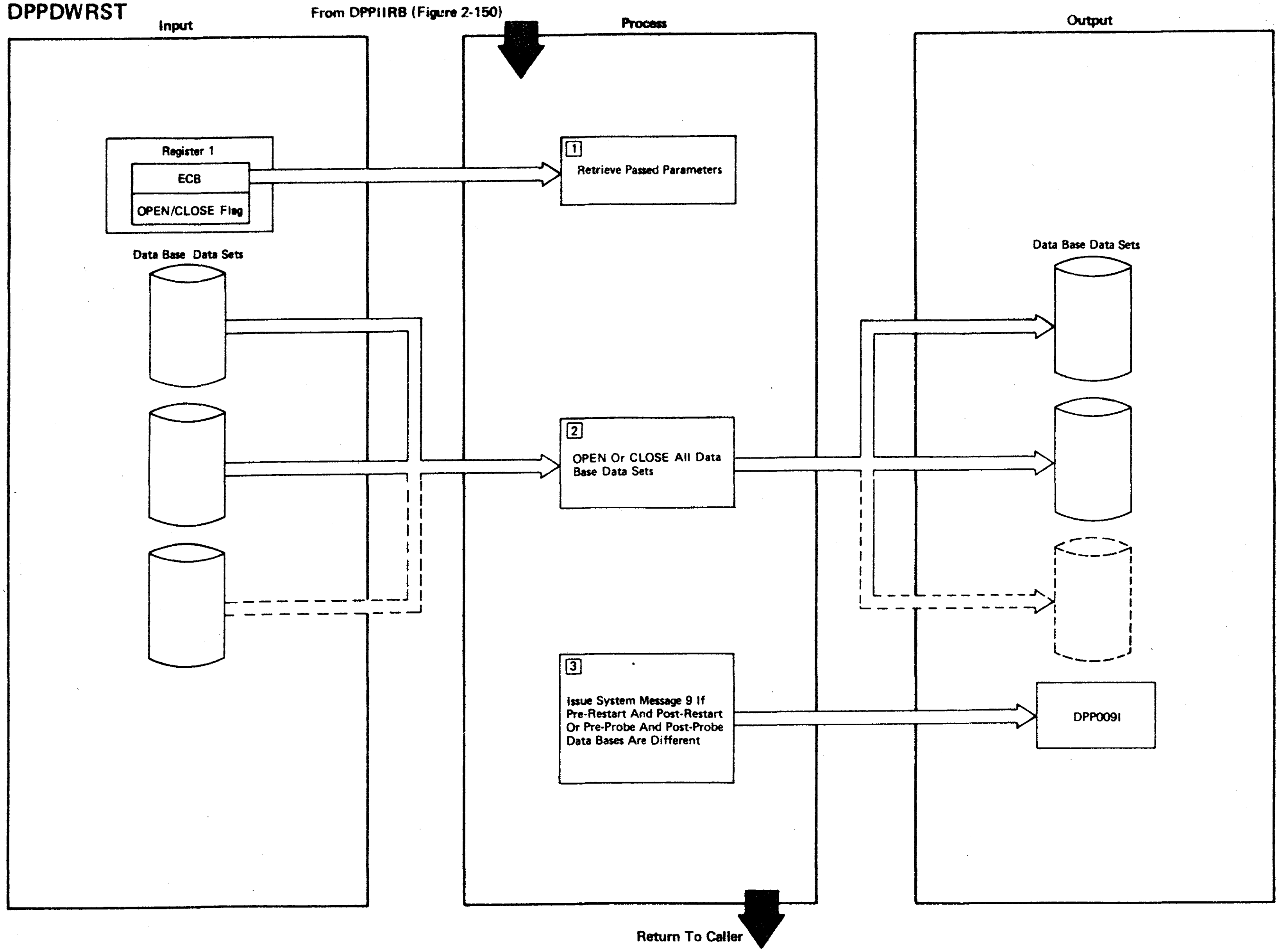

From DPPIIRB (Figure 2-150)
Input
Process
Output
Register 1
ECB
OPEN/CLOSE Flag
Data Base Data Sets
1
Retrieve Passed Parameters
2
OPEN Or CLOSE All Data Base Data Sets
3
Issue System Message 9 If Pre-Restart And Post-Restart Or Pre-Probe And Post-Probe Data Bases Are Different
Data Base Data Sets
DPP009I
Return To Caller

Figure 2-151 (1 Of 2) - Data Base OPEN/CLOSE

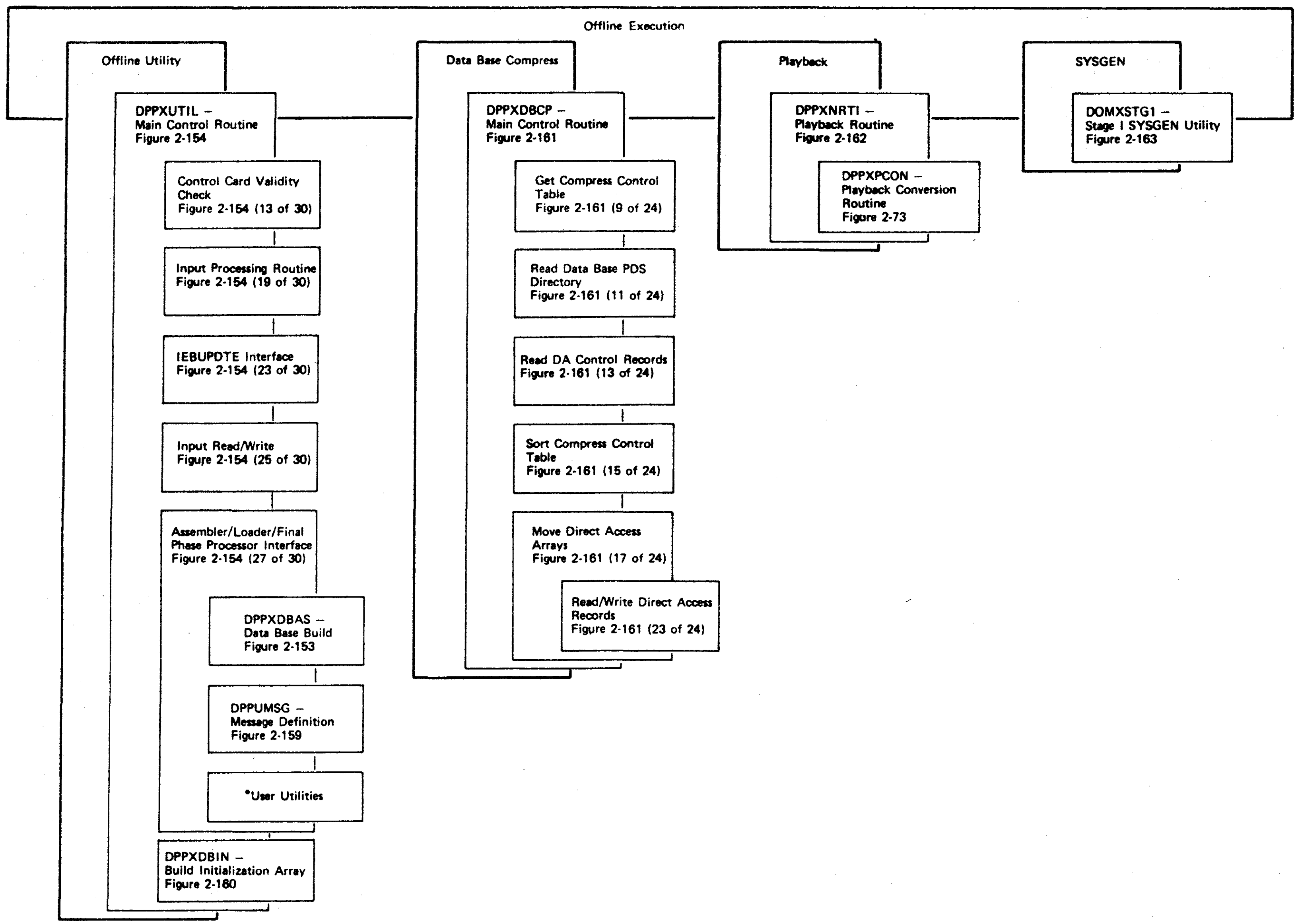
Figure 2-151 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	When DPPDWRST is entered, register 1 will point to a double word. The first word contains an ECB which is being waited on by DPPIIRB, and the second word contains an OPEN/CLOSE flag. If the second word is 1, all the data base data sets are opened, and if the second word is 0, all the data base data sets are closed.		DPPDWRST
2	All the data base data sets in the system are opened or closed according to the OPEN/CLOSE flag pointed to by register 1.		DPPDWRST
3	A BLDL will be executed to retrieve the @INIT data base directory entry. If the DBDIBUPD field in the @INIT entry is different from the DBDIBUPD field in the Data Base DDNAME Table, message 9 will be issued.	DPP0091	DPPDWRST

Intentionally Blank

Offline Utility

The offline utility is primarily designed to control the offline maintenance of the data base, message, and user-associated data sets used during online execution. Through an interface with the OS/VS1 utility, IEBUPDTE, source data sets, which serve as input to the online data set build, may be updated by the offline utility. Online data sets are updated through an interface which will invoke the appropriate final phase processor for the area (data base, messages, or user) being modified.


Offline Execution
Offline Utility
DPPXUTIL – Main Control Routine Figure 2-154
Control Card Validity Check Figure 2-154 (13 of 30)
Input Processing Routine Figure 2-154 (19 of 30)
IEBUPDTE Interface Figure 2-154 (23 of 30)
Input Read/Write Figure 2-154 (25 of 30)
Assembler/Loader/Final Phase Processor Interface Figure 2-154 (27 of 30)
DPPXDBAS – Data Base Build Figure 2-153
DPPUMSG – Message Definition Figure 2-159
*User Utilities
DPPXDBIN – Build Initialization Array Figure 2-160
Data Base Compress
DPPXDBCP – Main Control Routine Figure 2-161
Get Compress Control Table Figure 2-161 (9 of 24)
Read Data Base PDS Directory Figure 2-161 (11 of 24)
Read DA Control Records Figure 2-161 (13 of 24)
Sort Compress Control Table Figure 2-161 (15 of 24)
Move Direct Access Arrays Figure 2-161 (17 of 24)
Read/Write Direct Access Records Figure 2-161 (23 of 24)
Playback
DPPXNRTI – Playback Routine Figure 2-162
DPPXPCON – Playback Conversion Routine Figure 2-73
SYSGEN
DOMXSTG1 – Stage I SYSGEN Utility Figure 2-163

*User Associated Programs

Figure 2-152 - Special Real Time Operating System Offline Execution Overview

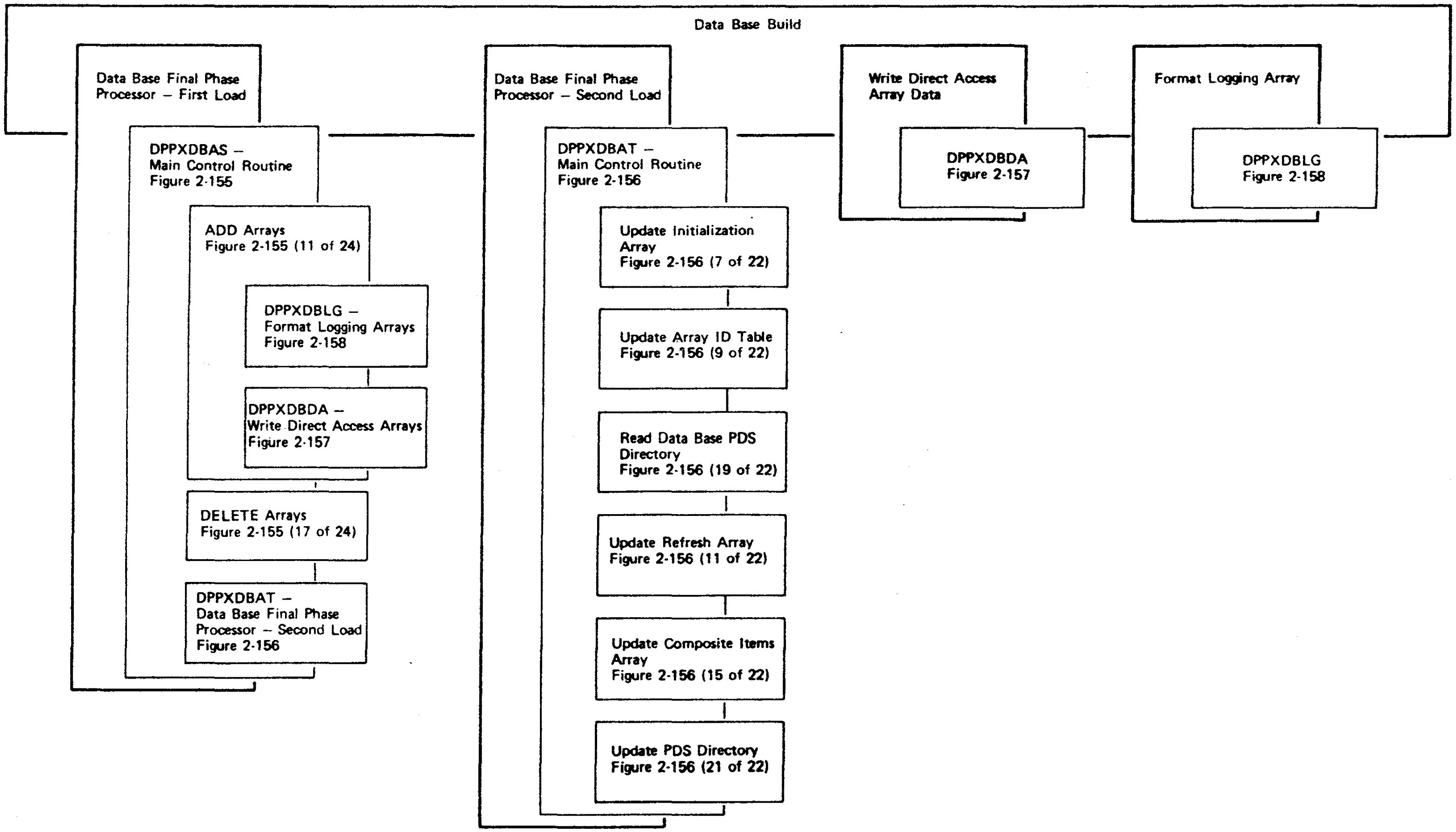

Data Base Build
Data Base Final Phase Processor – First Load
DPPXDBAS – Main Control Routine Figure 2-155
ADD Arrays Figure 2-155 (11 of 24)
DPPXDBLG – Format Logging Arrays Figure 2-158
DPPXDBDA – Write Direct Access Arrays Figure 2-157
DELETE Arrays Figure 2-155 (17 of 24)
DPPXDBAT – Data Base Final Phase Processor – Second Load Figure 2-156
Data Base Final Phase Processor – Second Load
DPPXDBAT – Main Control Routine Figure 2-156
Update Initialization Array Figure 2-156 (7 of 22)
Update Array ID Table Figure 2-156 (9 of 22)
Read Data Base PDS Directory Figure 2-156 (19 of 22)
Update Refresh Array Figure 2-156 (11 of 22)
Update Composite Items Array Figure 2-156 (15 of 22)
Update PDS Directory Figure 2-156 (21 of 22)
Write Direct Access Array Data
DPPXDBDA Figure 2-157
Format Logging Array
DPPXDBLG Figure 2-158

Figure 2-153 - Data Base Build Overview

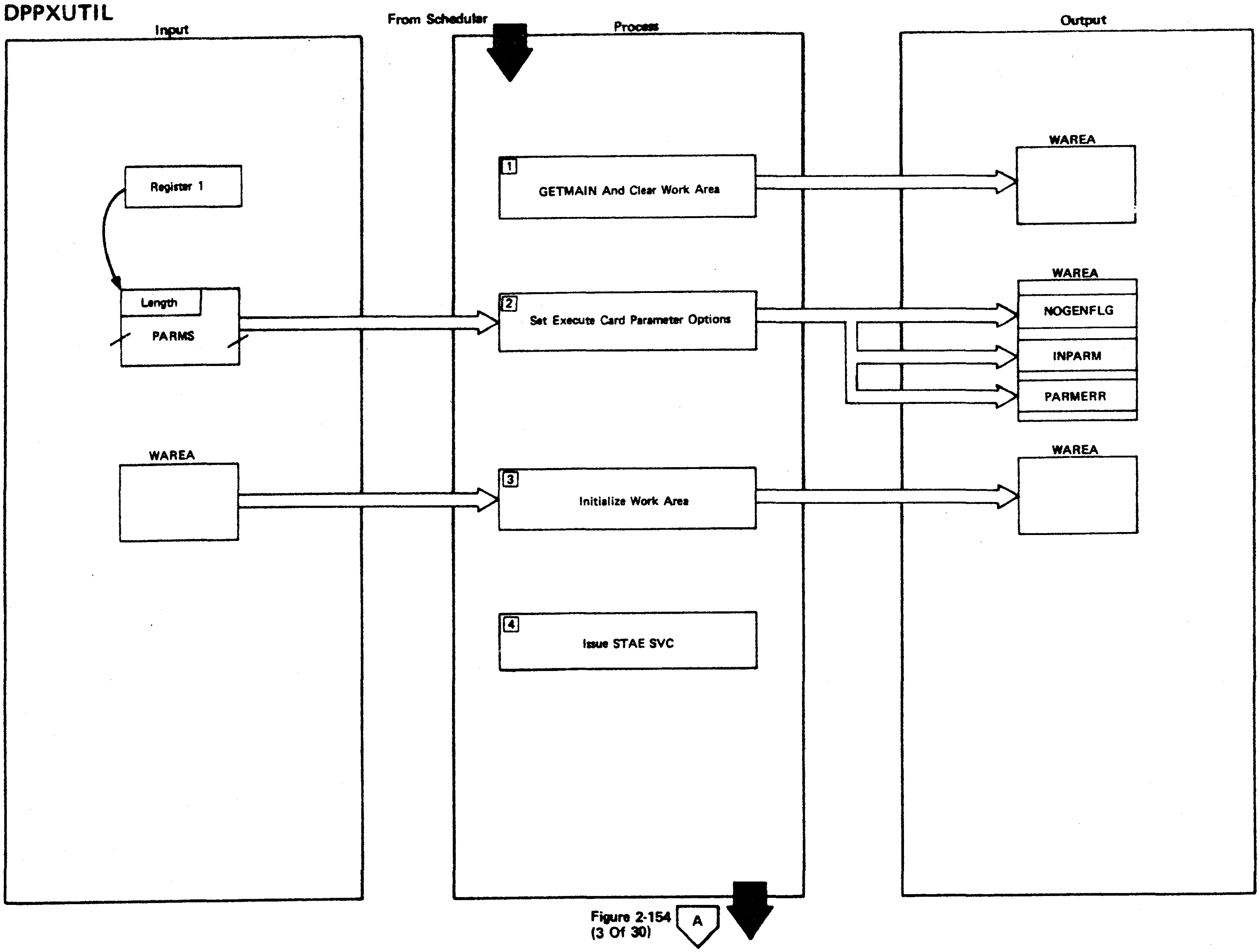


Figure 2-154 (1 Of 30) - Main Control Routine

Figure 2-154 (2 of 30).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	GETMAIN is used to obtain a storage area for register save area and for work space. The area is cleared to contain all binary zeros.		DPPXUTIL
2	The execute card parameters may specify the use of assembler H or the OS/VS1 assembler and may specify that input to the assembler be preceded by a PRINT GEN or a PRINT NOGEN. Default options are the OS/VS1 assembler and PRINT NOGEN. Specifying invalid options will result in setting the default options and a parameter error flag.		DPPXUTIL
3	Initialize read buffers, IEBUPDTE parm list, assembler parm list, loader parm list, sequential input DCB, and PDS input DCB areas of the work area.		DPPXUTIL
4	Issue STAE SVC to prepare for cleanup processing in case of an ABEND.		DPPXUTIL

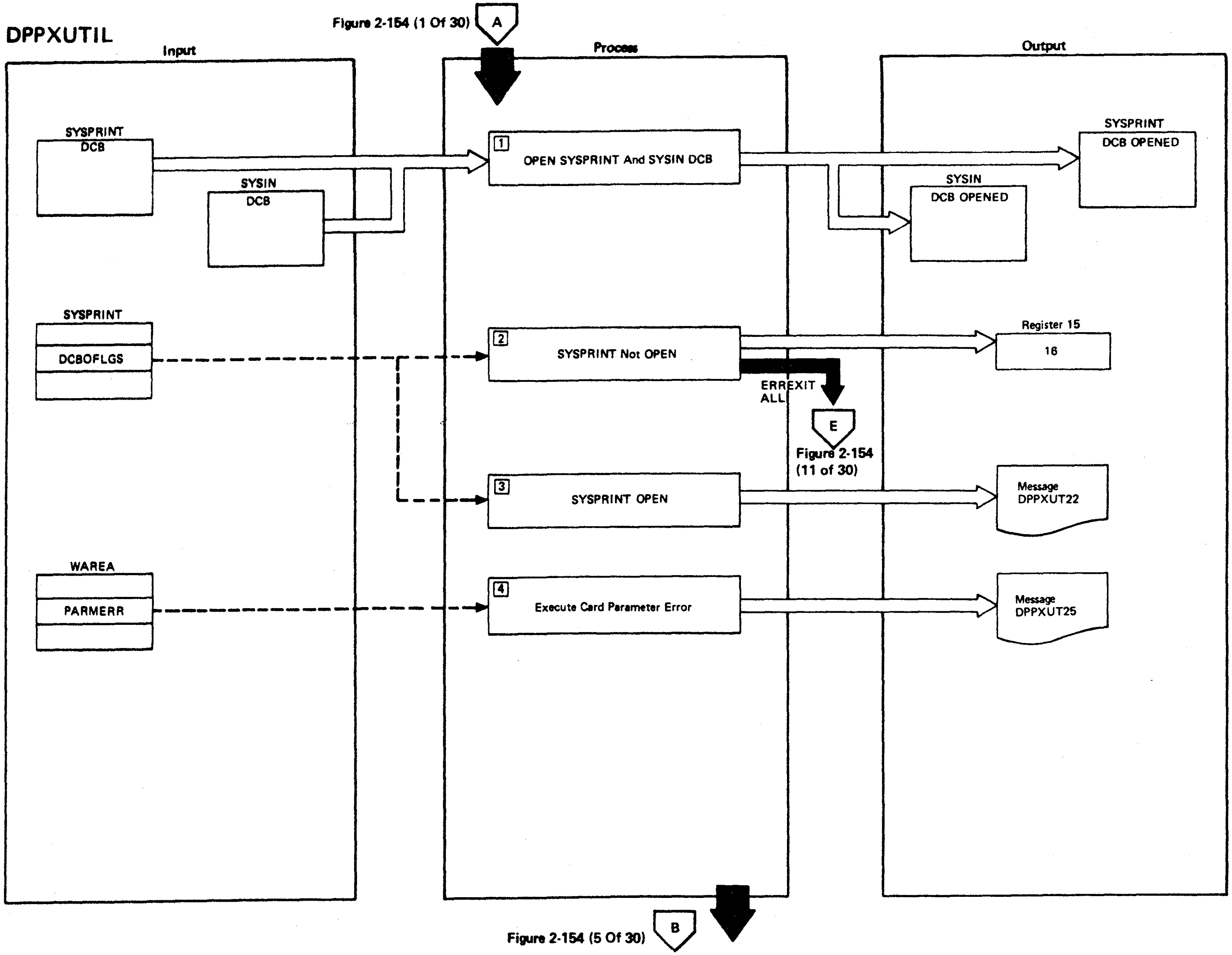


Figure 2-154 (3 Of 30) - Main Control Routine

Figure 2-154 (4 of 30).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The SYSPRINT and SYSIN data sets are opened.		DPPXUTIL
2	If SYSPRINT does not open, register 15 is set to 16, and error exit processing is performed.		DPPXUTIL
3	If SYSPRINT is open, the DPPXUTIL header message is written to SYSPRINT.	DPPXUT22I	DPPXUTIL
4	If errors were found in the execute card parameters, a message indicating that default options are used is written to SYSPRINT.	DPPXUT25I	DPPXUTIL

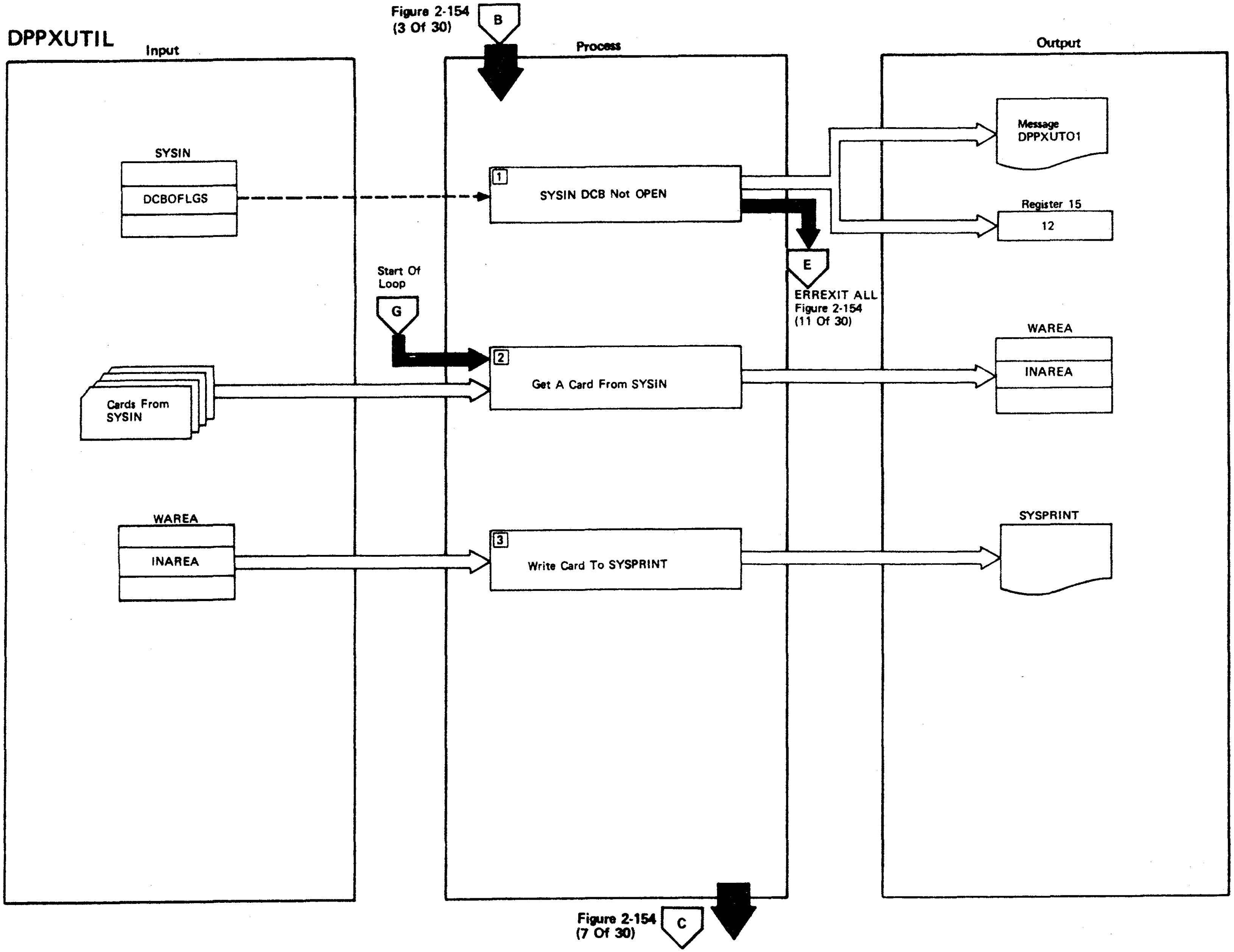


Figure 2-154 (5 Of 30) - Main Control Routine

Figure 2-154 (6 of 30).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the SYSIN DCB is not open, a missing DD statement message is written to SYSPRINT, return code 12 is set in register 15, and error exit processing is performed.	DPPXUT01I	DPPXUTIL
2	The input buffer is checked for a control card. If no control card is present, a GET is issued to obtain the first card from SYSIN. When the end-of-file is read from SYSIN, the message SYSIN END-OF-FILE is written to SYSPRINT.	DPPXUT20I	DPPXUTIL
3	The card now residing in the input buffer is written to SYSPRINT.		DPPXUTIL

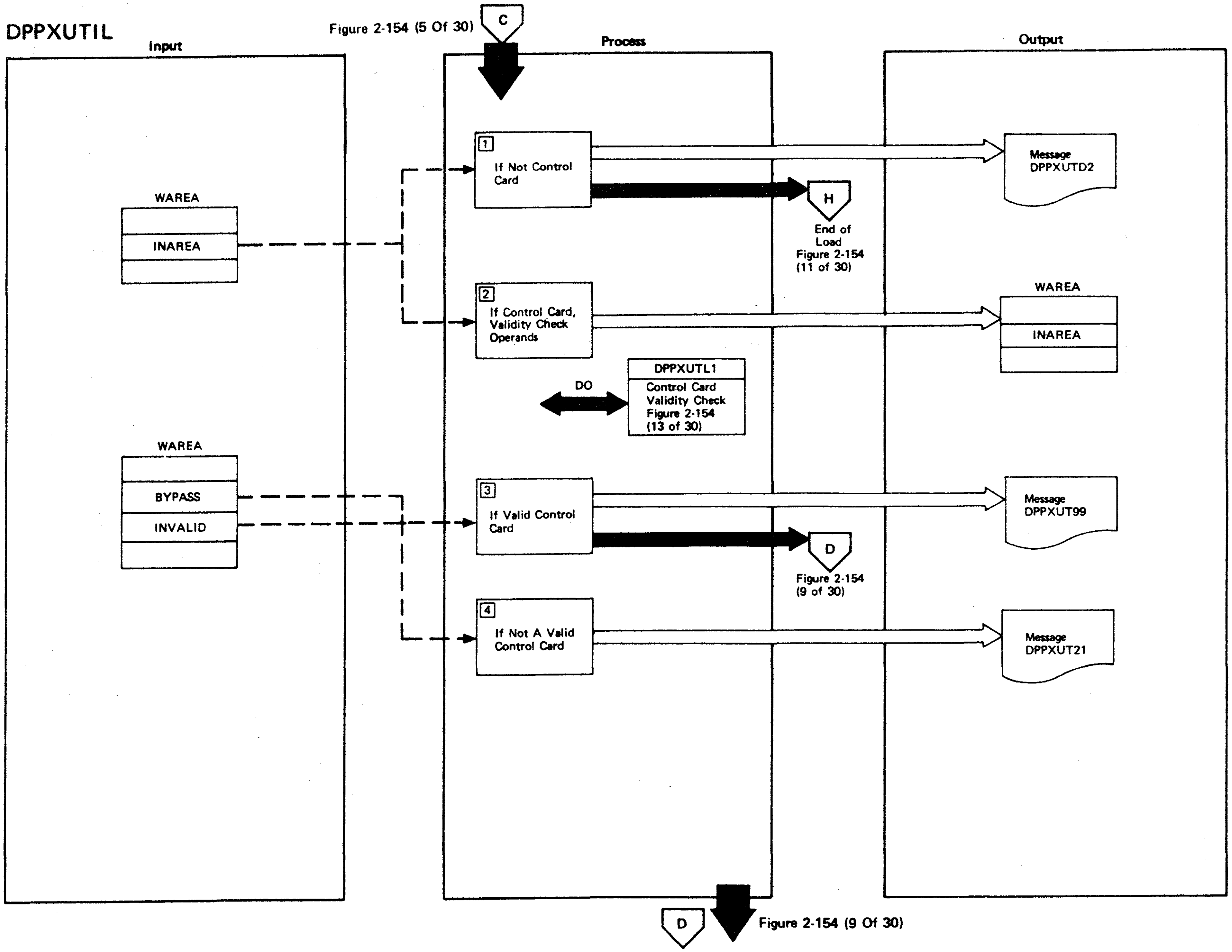


Figure 2-154 (7 Of 30) - Main Control Routine

Figure 2-154 (8 of 30).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the card in the input buffer is not a control card, print a message that the first card must be a control card. Read and flush cards from SYSIN until a control card or an end-of-file is reached.	DPPXUT02I	DPPXUTIL
2	If the card in the input buffer is a control card, go to the control card validity check routine.		DPPXUTIL
3	If the control card is valid, the CONTROL CARD ACCEPTED message is written to SYSPRINT.	DPPXUT99I	DPPXUTIL
4	If the control card is not valid, the SKIPPING FOR NEXT CONTROL CARD message is written to SYSPRINT.	DPPXUT21I	DPPXUTIL

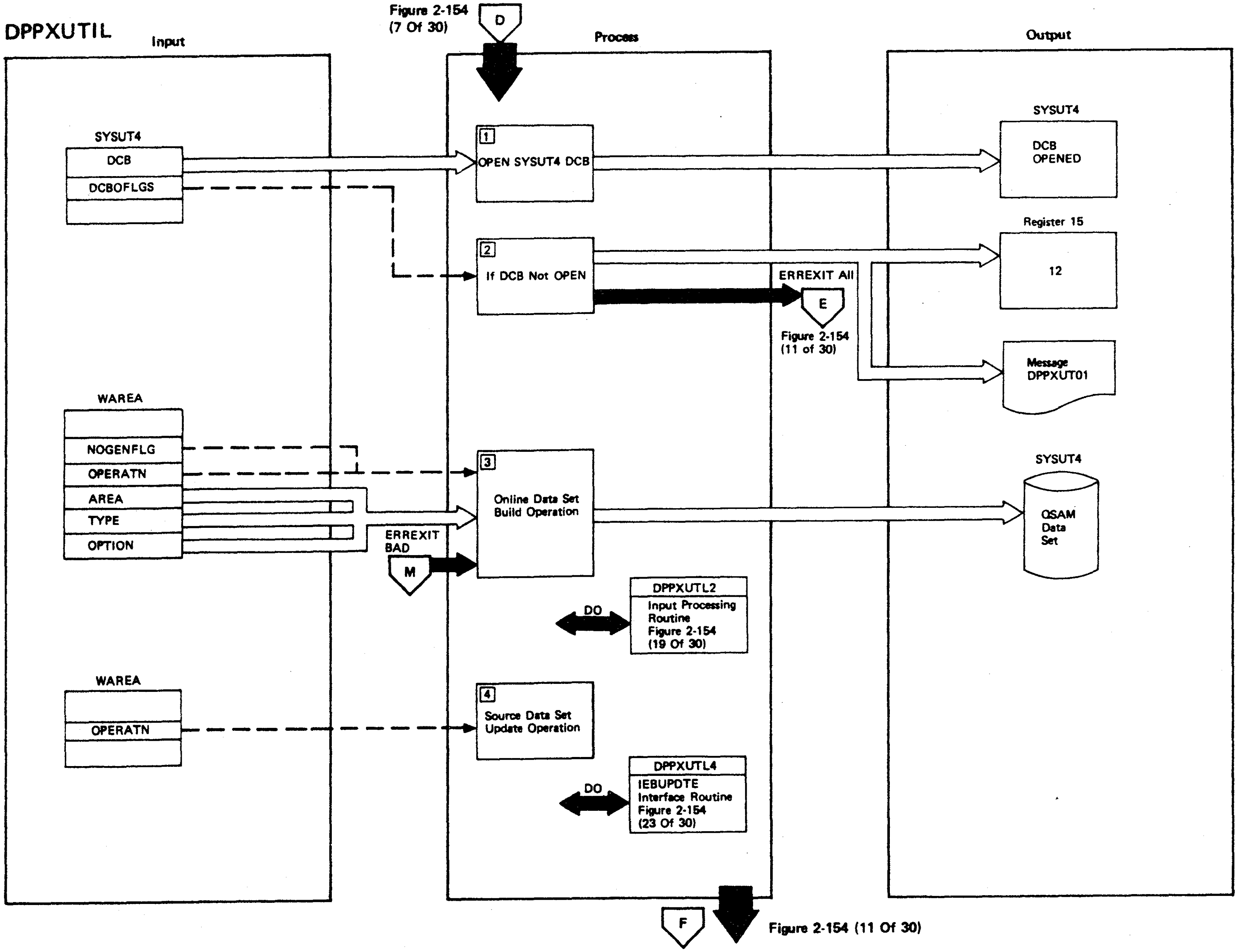


Figure 2-154 (9 Of 30) · Main Control Routine

Figure 2-154 (10 of 30).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The utility data set described by the SYSUT4 DD statement is opened.		DPPXUTIL
2	If the SYSUT4 DCB does not open, a missing DD statement message is written to SYSPRINT, return code 12 is set in register 15, and error exit processing is performed.	DPPXUT01I	DPPXUTIL
3	If the NOGENFLG is set, a PRINT NOGEN card is written to SYSUT4. An area definition macro card is constructed from the input control card and written to SYSUT4. If the operation is an online data set build, control is passed to the input processing control routine for online data set build.		DPPXUTIL
4	If the operation is a source data set update, control is passed to the IEBUPDTE interface routine for source data set update.		DPPXUTIL

DPPXUTIL

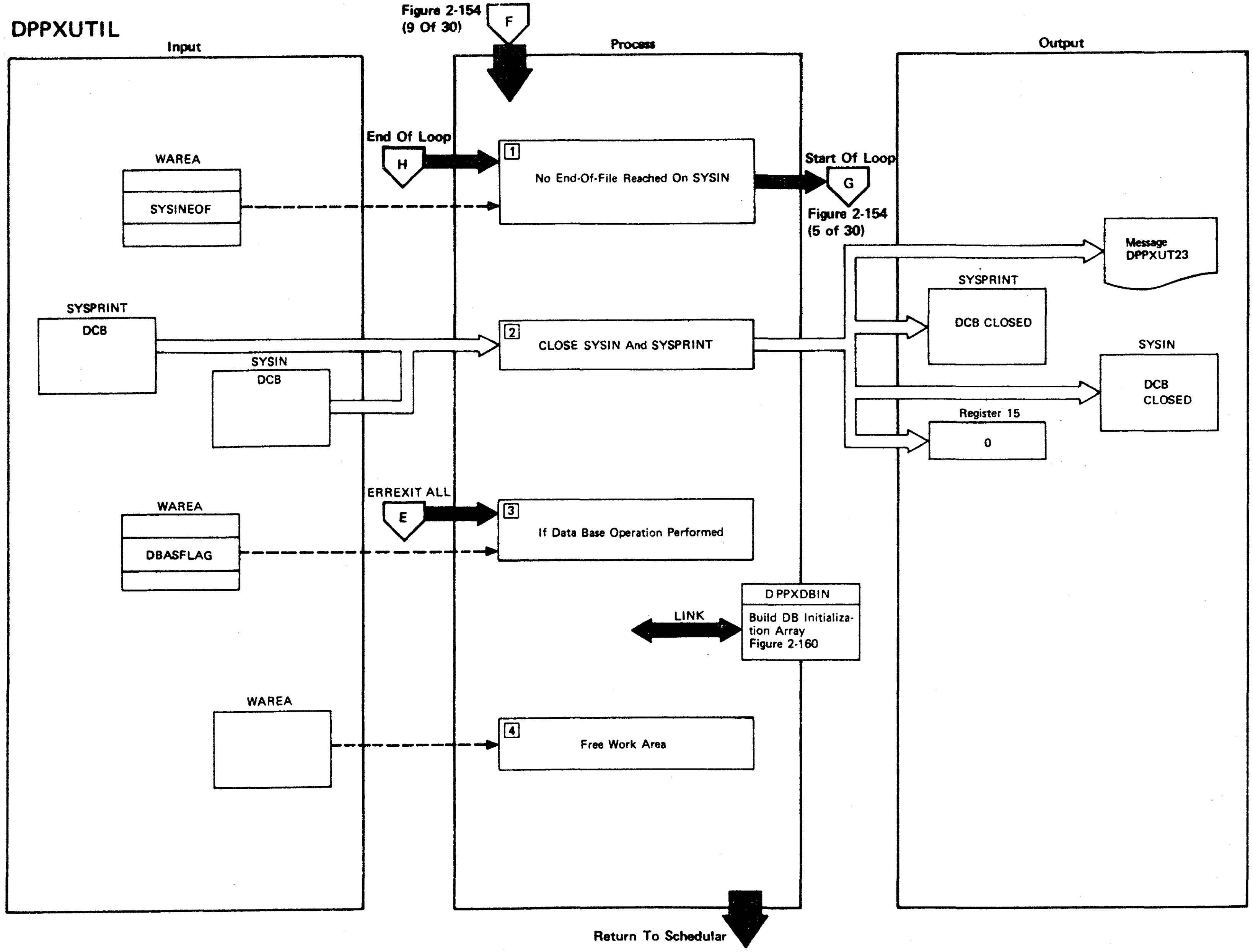


Figure 2-154 (11 Of 30) - Main Control Routine

Figure 2-154 (12 of 30).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If end-of-file has not been reached on SYSIN, go back and process next control card.		DPPXUTIL
2	If end-of-file has been reached on SYSIN, write the UTILITY ENDED message to SYSPRINT, CLOSE SYSPRINT and SYSIN, and set return code 0 in register 15.	DPPXUT23I	DPPXUTIL
3	If a data base operation was performed, link to DPPXDBIN to build the data base initialization array, @INIT.		DPPXUTIL
4	Free the gotten work area.		DPPXUTIL

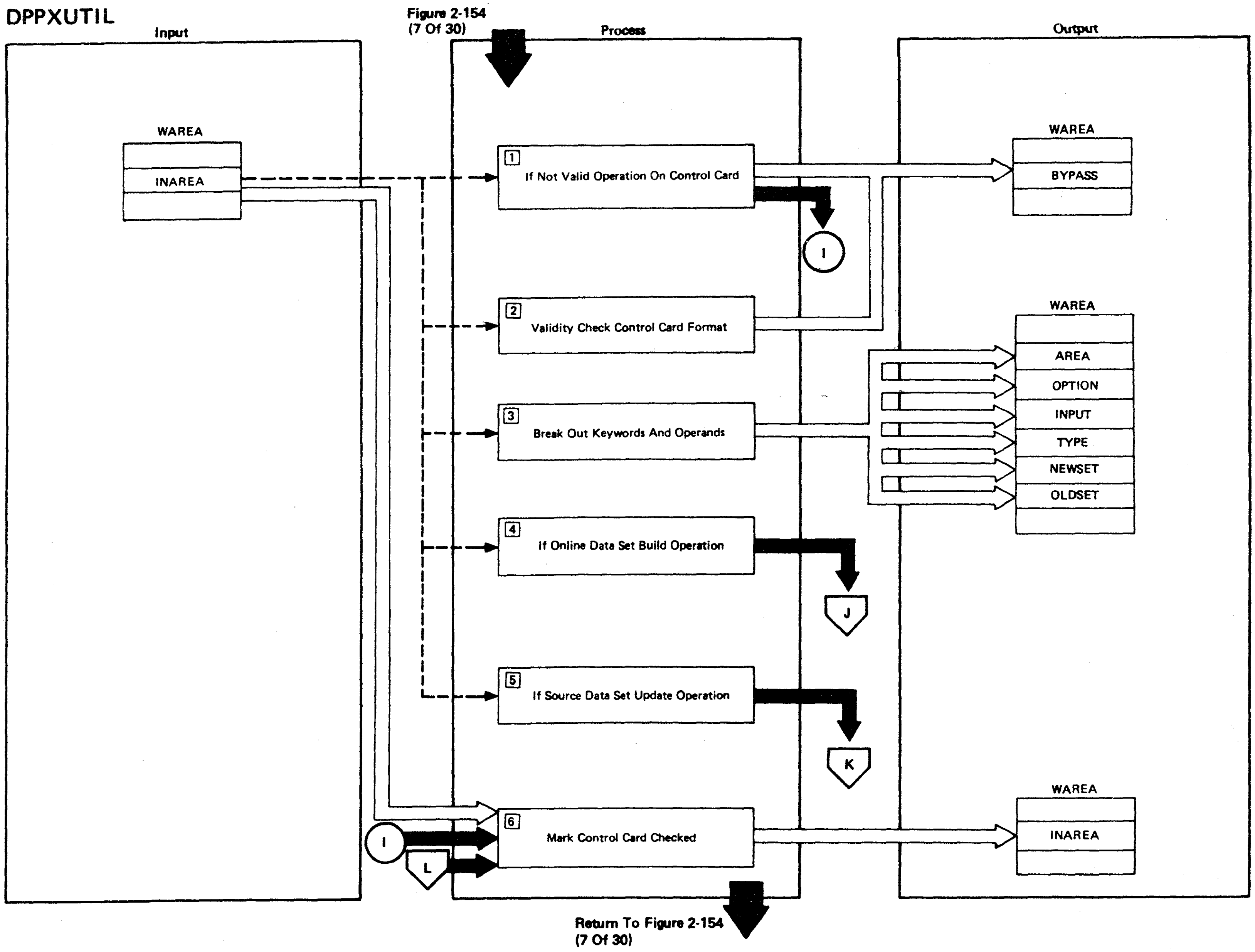


Figure 2-154
(7 Of 30)
Input
Process
Output
WAREA
INAREA
1 If Not Valid Operation On Control Card
2 Validity Check Control Card Format
3 Break Out Keywords And Operands
4 If Online Data Set Build Operation
5 If Source Data Set Update Operation
6 Mark Control Card Checked
I
J
K
I
L
WAREA
BYPASS
WAREA
AREA
OPTION
INPUT
TYPE
NEWSET
OLDSET
WAREA
INAREA
Return To Figure 2-154
(7 Of 30)

Figure 2-154 (13 Of 30) - DPPXUTL1 - Control Card Validity Check

Figure 2-154 (14 of 30).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The control card is checked for a valid operation field. A message is written to SYSPRINT if the operation is omitted or incorrect. Set bypass if message written to SYSPRINT.	DPPXUT18I DPPXUT19I	DPPXUTIL
2	The control card is checked for proper format and continuation. Messages are written to SYSPRINT for:		DPPXUTIL
	PARAMETER OR CONTINUATION MARK MISSING	DPPXUT03I	
	EXPECTED CONTINUATION NOT RECEIVED	DPPXUT04I	
	COLUMNS 1-15 MUST BE BLANK	DPPXUT05I	
	CONTROL CARD TEXT BEYOND COL 71.	DPPXUT06I	
	Set bypass if message written to SYSPRINT.		DPPXUTIL
3	Break out keyword operands into separate fields.		DPPXUTIL
4	Validity check keywords and operands on online data set build control card.		DPPXUTIL
5	Validity check keywords and operands on source data set update control card.		DPPXUTIL
6	Mark control card to indicate completion of validity checking.		DPPXUTIL

DPPXUTIL

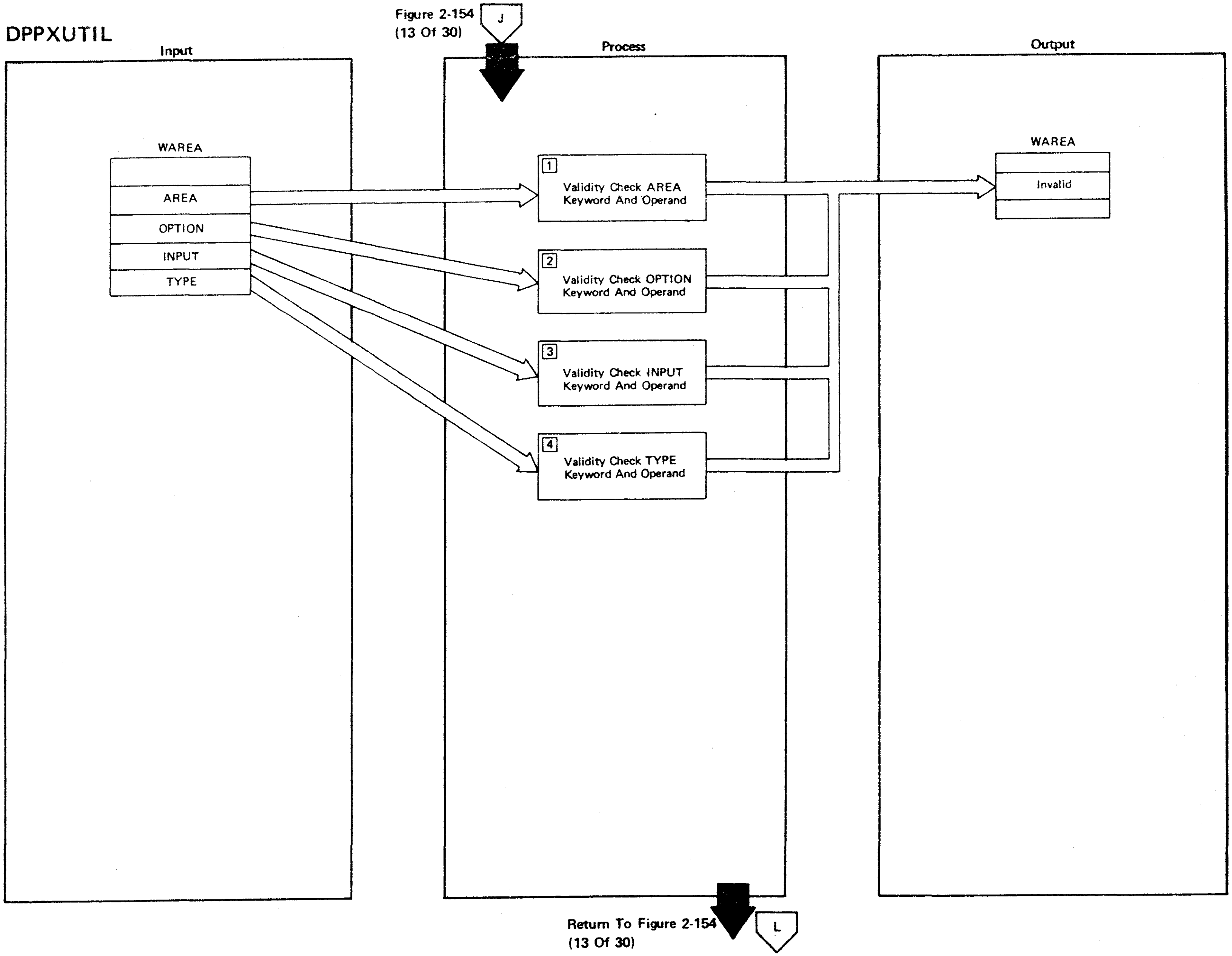


Figure 2-154
(13 Of 30)
J
Input
Process
Output
WAREA
AREA
OPTION
INPUT
TYPE
1
Validity Check AREA Keyword And Operand
2
Validity Check OPTION Keyword And Operand
3
Validity Check INPUT Keyword And Operand
4
Validity Check TYPE Keyword And Operand
WAREA
Invalid
Return To Figure 2-154
(13 Of 30)
L

Figure 2-154 (15 Of 30) - DPPXUTL1 - Control Card Validity Check

Figure 2-154 (16 of 30).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The operand specified for AREA is validity checked.*		DPPXUTIL
2	The operand specified for OPTION is validity checked.*		DPPXUTIL
3	The operand specified for INPUT is validity checked.*		DPPXUTIL
4	The operand specified for TYPE is validity checked.*		DPPXUTIL
	*Messages written as a result of this validity checking are:		
	WRONG PARAMETER	DPPXUT07I	
	MULTIPLE KEYWORD	DPPXUT08I	
	PARAMETER IN ERROR	DPPXUT09I	
	RIGHT PARENTHESIS MISSING, TREATED AS VALID	DPPXUT10I	
	WRONG KEYWORD	DPPXUT11I	
	INPUT SPECIFICATION MISSING	DPPXUT12I	
	AREA SPECIFICATION MISSING	DPPXUT13I	
	NO OPERAND FOUND	DPPXUT17I	

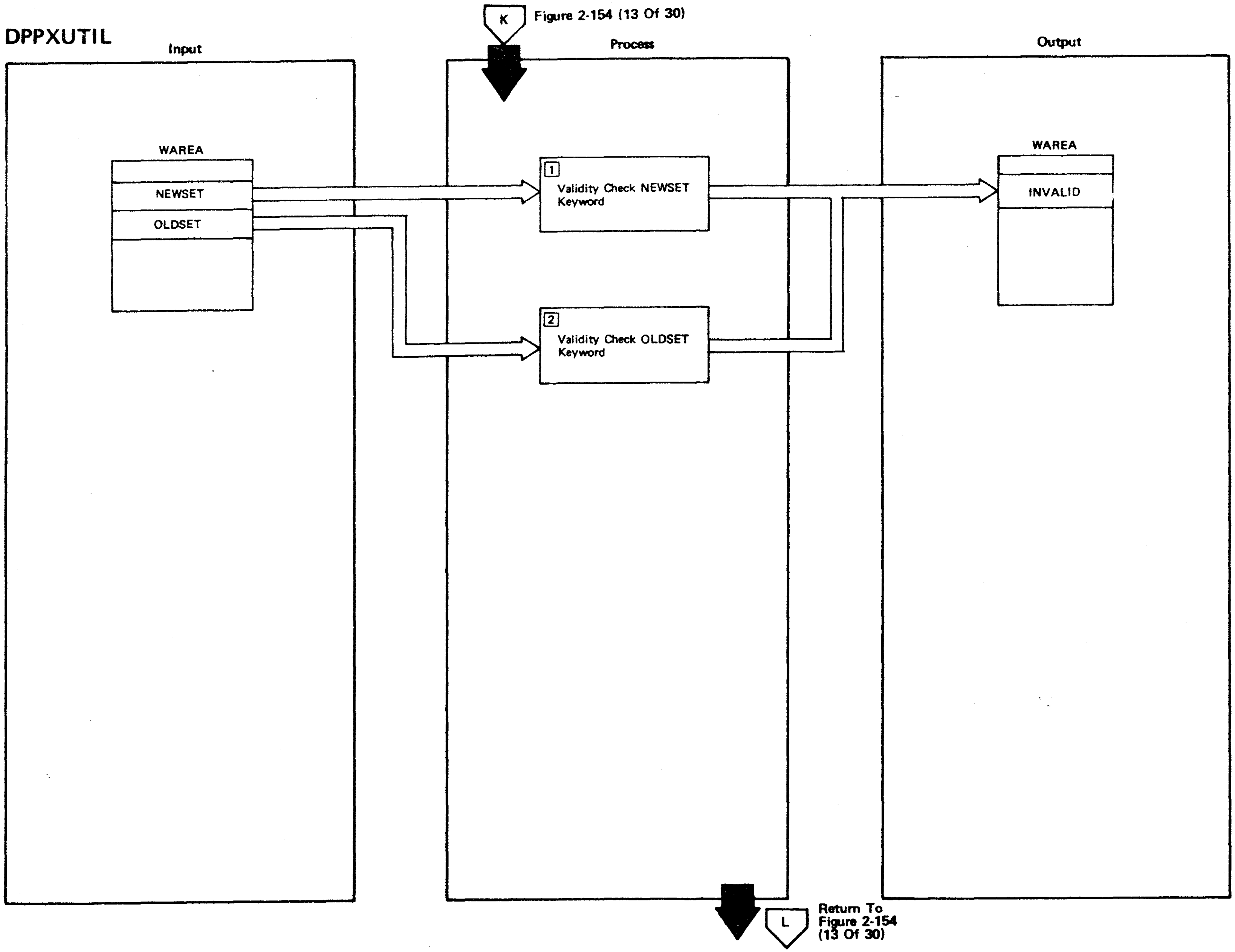

K
Figure 2-154 (13 Of 30)
Input
Process
Output
WAREA
NEWSET
OLDSET
1
Validity Check NEWSET Keyword
2
Validity Check OLDSET Keyword
WAREA
INVALID
L
Return To Figure 2-154 (13 Of 30)

Figure 2-154 (17 Of 30) - DPPXUTL1 - Control Card Validity Check

Figure 2-154 (18 of 30).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The operand specified for NEWSET is validity checked.*		DPPXUTIL
2	The operand specified for OLDSET is validity checked.*		DPPXUTIL
	*Messages written as a result of this validity checking are:		
	WRONG PARAMETER	DPPXUT07I	
	MULTIPLE KEYWORD	DPPXUT08I	
	PARAMETER IN ERROR	DPPXUT09I	
	WRONG KEYWORD	DPPXUT11I	
	NEWSET SPECIFICATION MISSING	DPPXUT15I	
	OLDSET SPECIFICATION MISSING	DPPXUT16I	

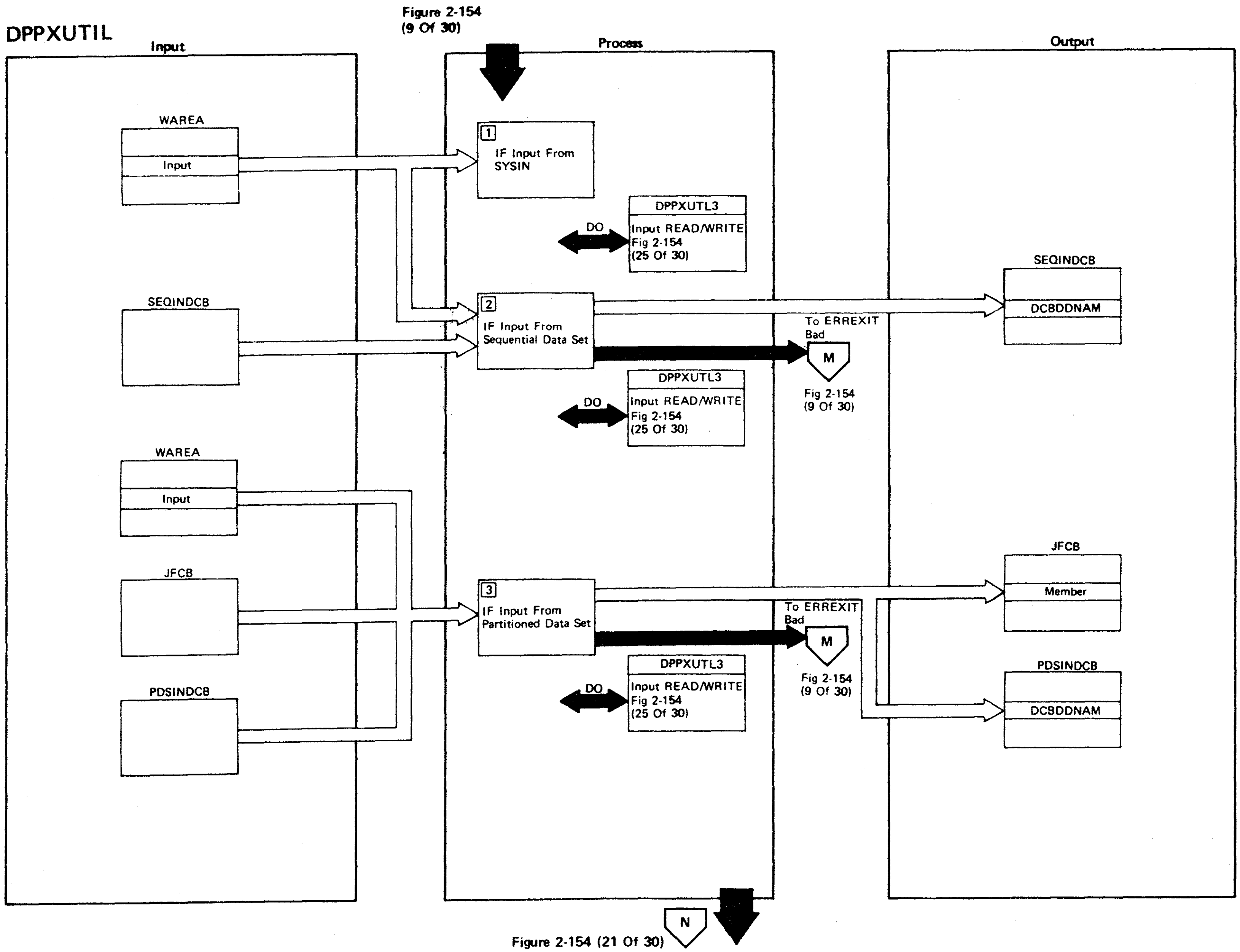


Figure 2-154 (19 Of 30) - DPPXUTL2 - Input Processing Control

Figure 2-154 (22 of 30).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Perform external interfaces with assembler, loader, and final phase processor.		DPPXUTIL

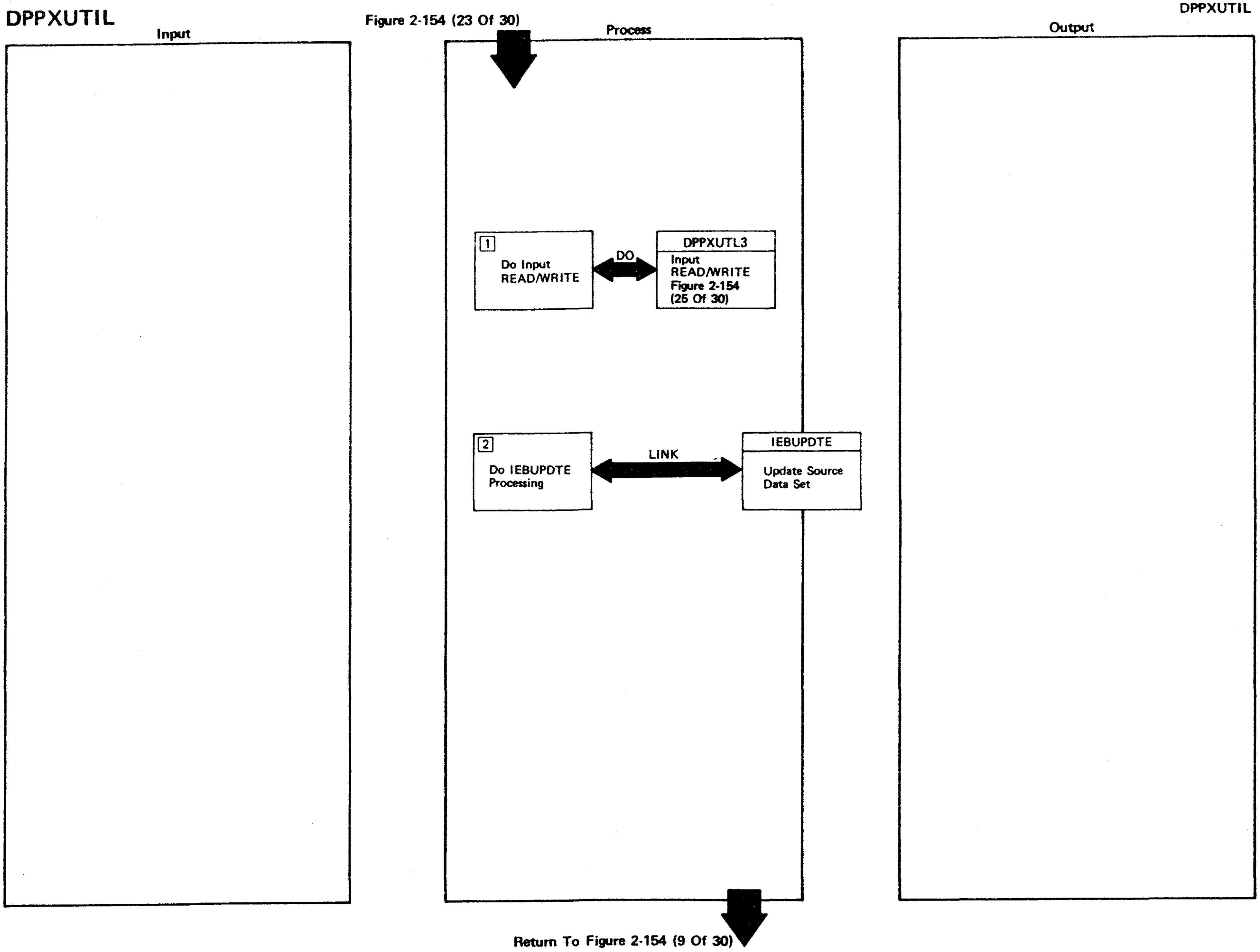


Figure 2-154 (23 Of 30) - DPPXUTL4 - IEBUPDTE Interface Routine

Figure 2-154 (24 of 30).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Control is passed to the input READ/WRITE routine.		DPPXUTIL
2	Link to IEBUPDTE to update the source data set.		DPPXUTIL

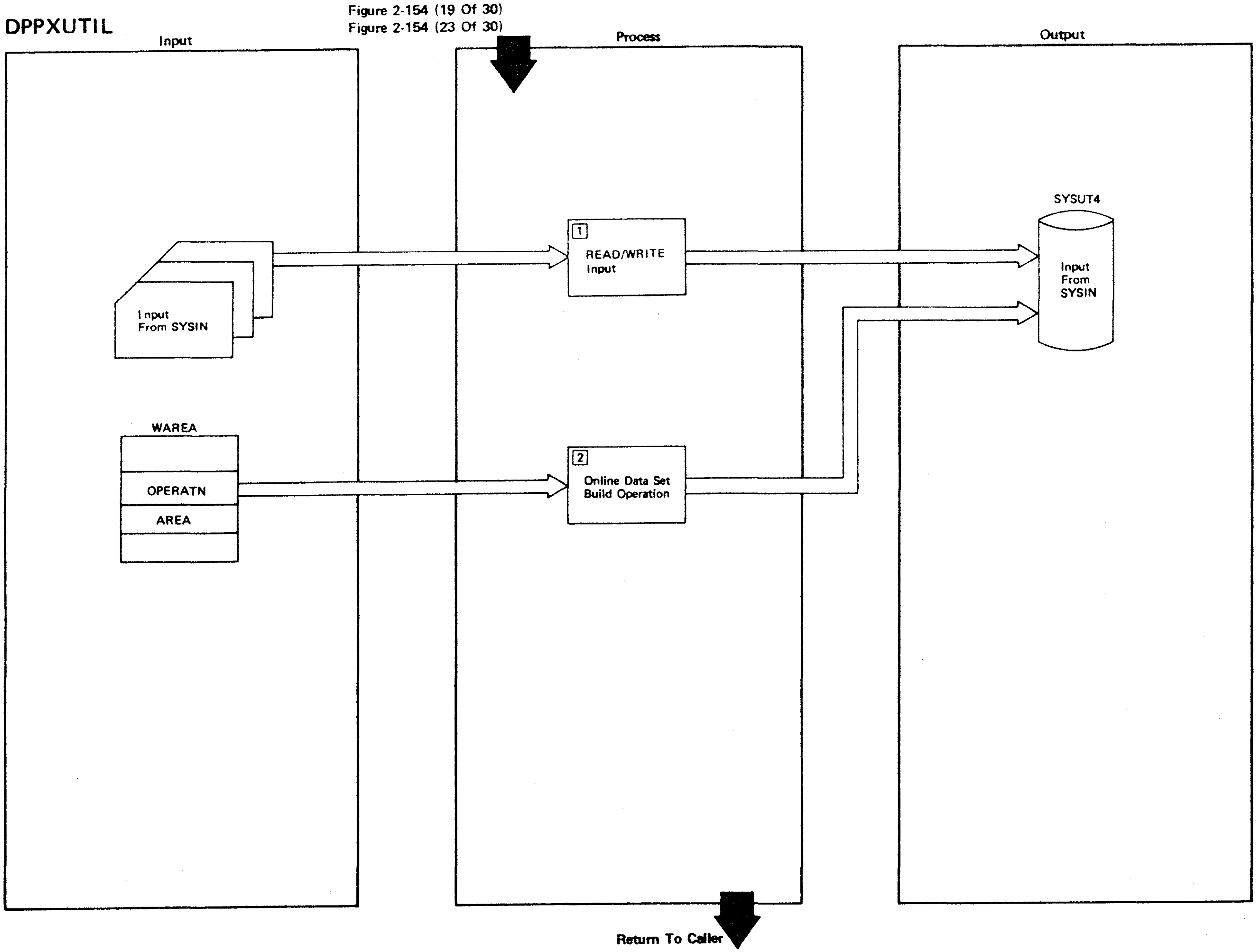


Figure 2-154 (25 Of 30) - DPPXUTL3 - Input Read/Write

Figure 2-154 (26 of 30).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	READ input from SYSIN and WRITE input cards to SYSUT4. The message END-OF-FILE ON INPUT DATA SET is written to SYSPRINT when the end-of-file is read on the input data set.	DPPXUT24I	DPPXUTIL
2	If online data set build operation, write appropriate area definition END macro to SYSUT4.		DPPXUTIL

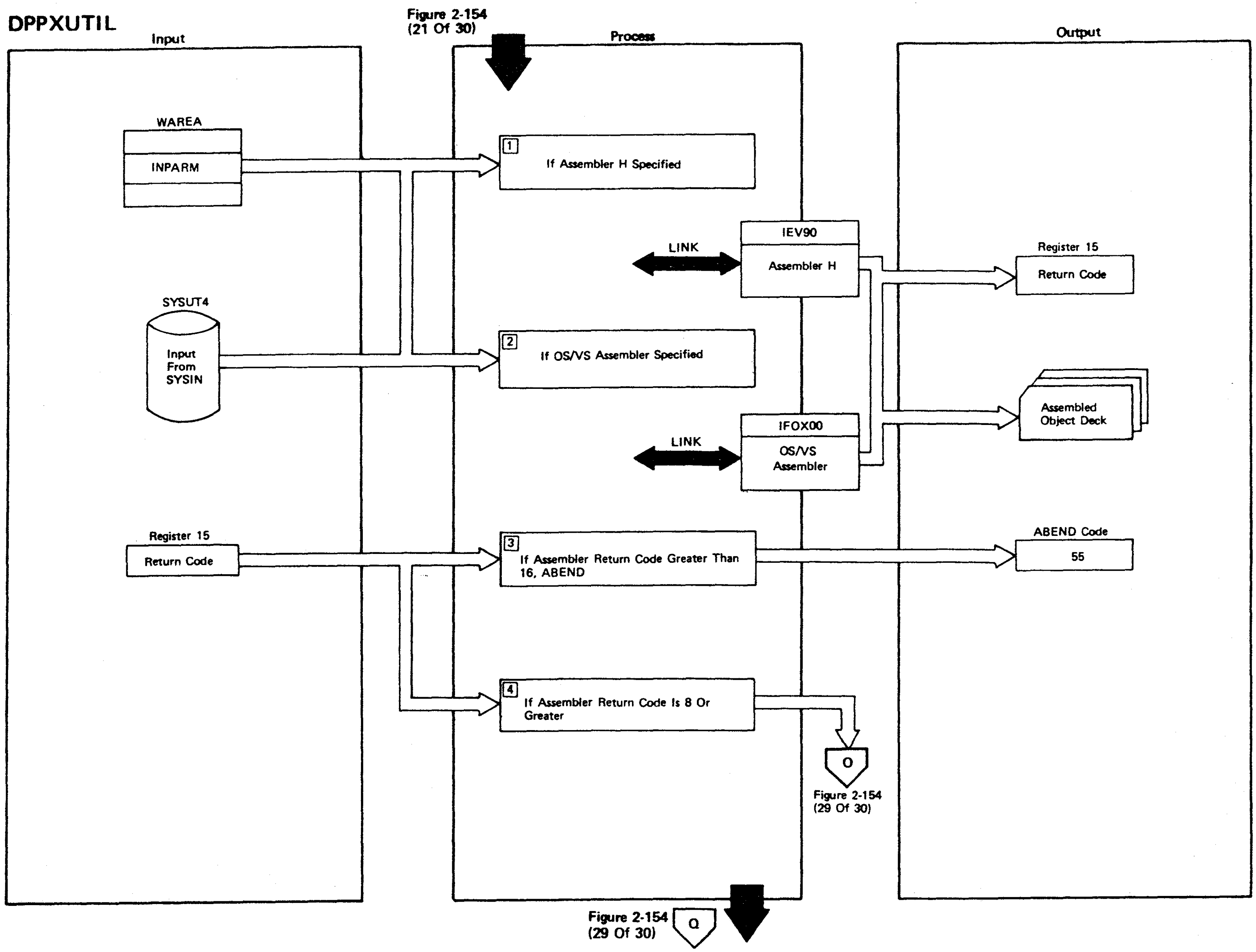


Figure 2-154 (27 Of 30) - DPPXUTL5 - ASM/Load/FPP Interface

Figure 2-154 (28 of 30).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If Assembler H was specified, the input cards on SYSUT4 are assembled by the Assembler H.		DPPXUTIL
2	If the OS/VS1 Assembler is specified, the input cards on SYSUT4 are assembled by the OS/VS1 Assembler.		DPPXUTIL
3	If the assembler return code in register 15 is greater than 16, DPPXUTIL ABENDs with ABEND code 55.	USER 55	DPPXUTIL
4	If the assembler return code in register 15 is 8 or greater, the PROCESSING ABORTED DUE TO BAD RETURN CODE FROM ASSEMBLER message is written to SYSPRINT.	DPPXUT26I	DPPXUTIL

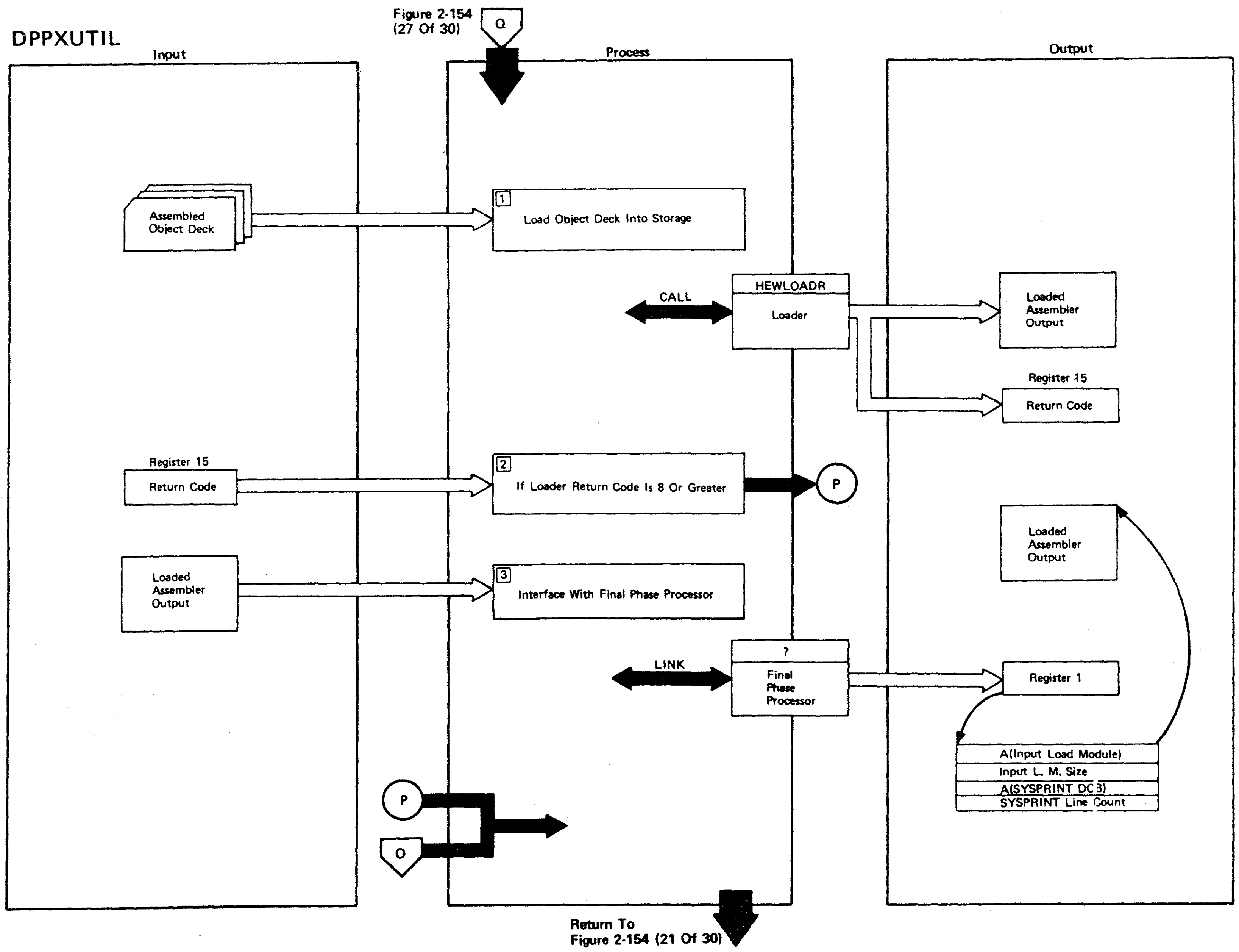


Figure 2-154 (29 Of 30) - DPPXUTL5 - ASM/Load/FPP Interface

Figure 2-154 (30 of 30).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The assembled object deck is loaded into storage by the LOADER. A return code is returned in register 15.		DDPXUTIL
2	If the return code is 8 or greater, the PROCESSING ABORTED DUE TO BAD RETURN CODE FROM LOADER message is written to SYSPRINT.	DPPXUT26I	DPPXUTIL
3	A link is performed to the final phase processor for online data set build. The name of the final phase processor is not known. The address of a program name is contained in the loaded assembler output.		DPPXUTIL

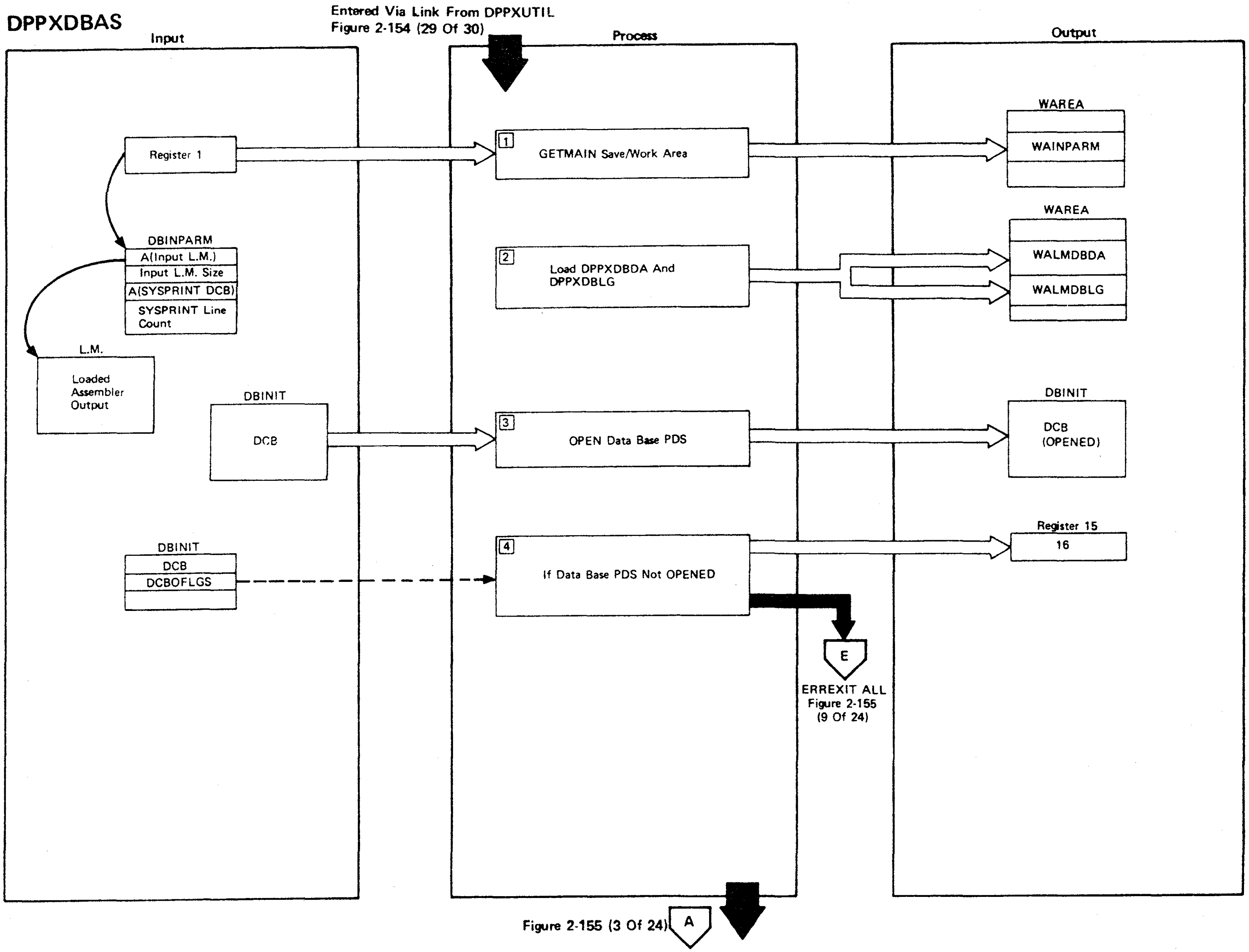


Figure 2-155 (1 Of 24) - Main Control Routine

Figure 2-155 (2 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	GETMAIN is used to obtain a register save area and work area. The address of the input parameter list is saved in the work area.		DPPXDBAS
2	The data base final phase processor branch routines are LOADED into storage.		DPPXDBAS
3	The DATA BASE FINAL PHASE PROCESSOR ENTERED message is written to SYSPRINT and the data base PDS is OPENed.	DPPXDB01I	DPPXDBAS
4	If the OPEN fails, the UNABLE TO OPEN DATA BASE PDS message is written to SYSPRINT, register 15 is set to 16, and error exit processing is performed.	DPPXDB07I	DPPXDBAS

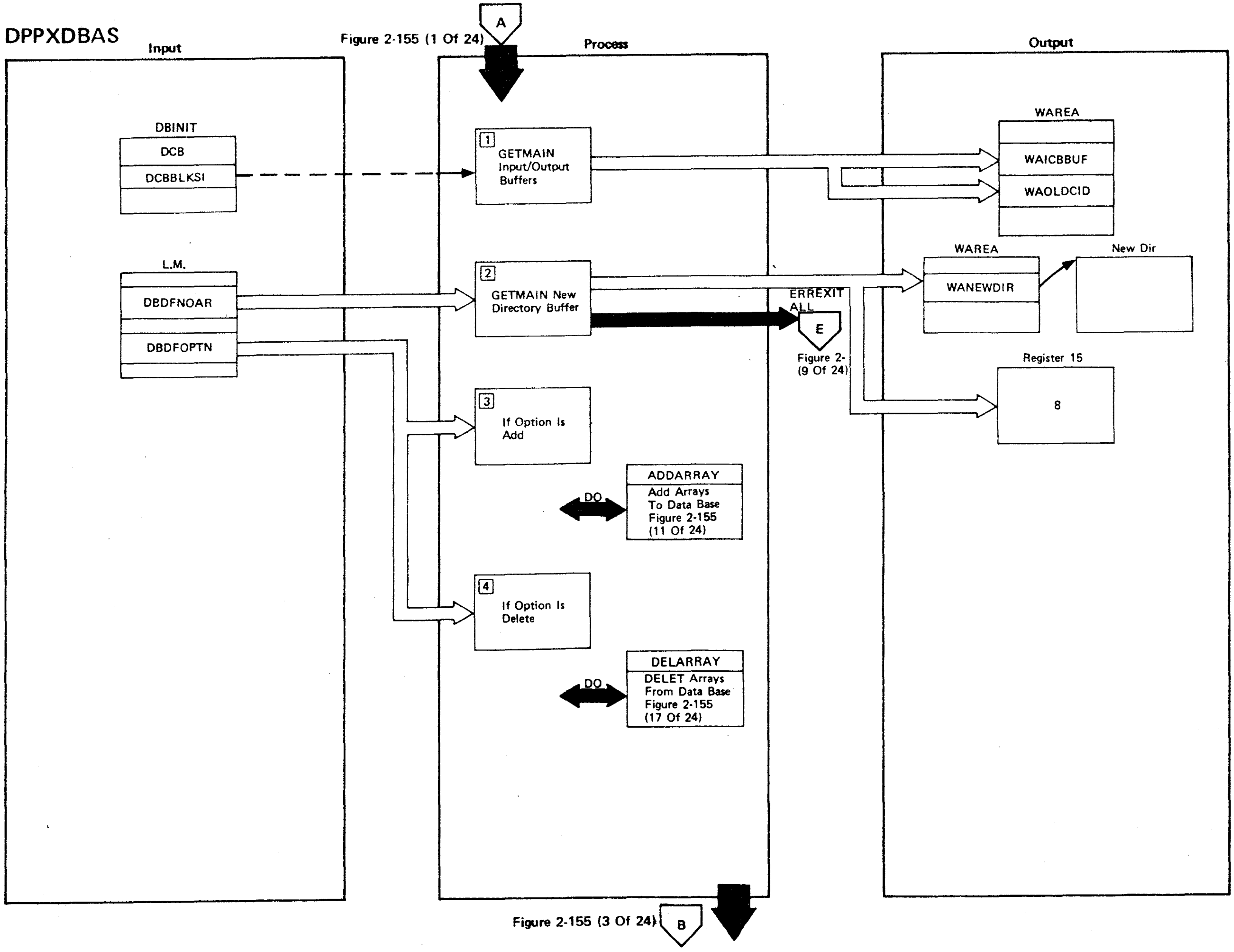


Figure 2-155 (3 Of 24) - Main Control Routine

Figure 2-155 (4 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	GETMAIN is used to obtain buffers for input and output. The addresses of the buffers are saved in the work area.		DPPXDBAS
2	If arrays have been defined in the input load module, GETMAIN is used to obtain storage for new directory entries. Otherwise, the message NO ARRAYS DEFINED is written to SYSPRINT, register 15 is set to return code 8, and error exit processing is performed.	DPPXDB05I	DPPXDBAS
3	If the processing option is ADD, go to the ADD array routine.		DPPXDBAS
4	If the processing option is DELETE, go to the DELETE array routine.		DPPXDBAS

DPPXDBAS

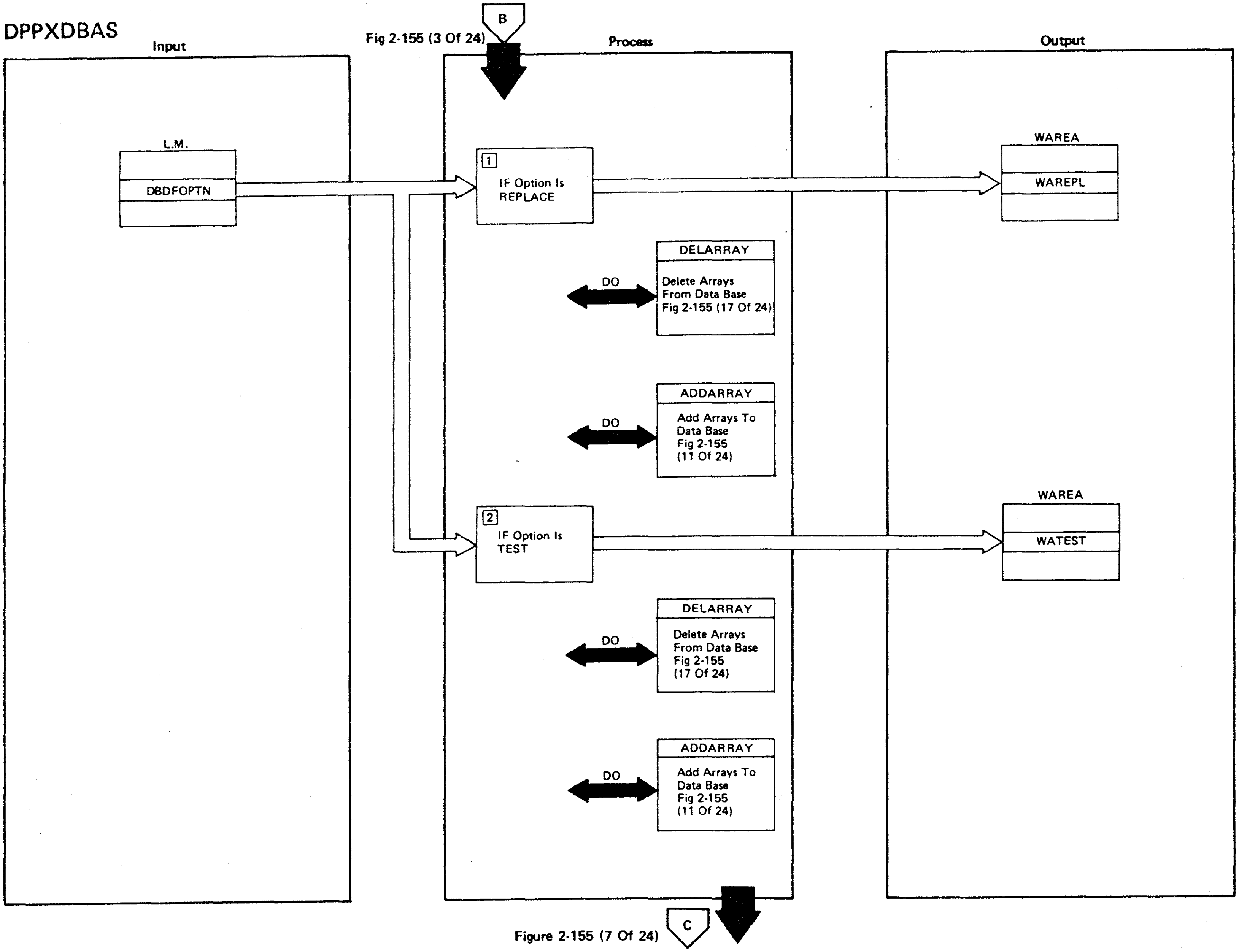

Input
Process
Output
B
Fig 2-155 (3 Of 24)
L.M.
DBDFOPTN
1
IF Option Is REPLACE
WAREA
WAREPL
DELARRAY
DO
Delete Arrays From Data Base Fig 2-155 (17 Of 24)
ADDARRAY
DO
Add Arrays To Data Base Fig 2-155 (11 Of 24)
2
IF Option Is TEST
WAREA
WATEST
DELARRAY
DO
Delete Arrays From Data Base Fig 2-155 (17 Of 24)
ADDARRAY
DO
Add Arrays To Data Base Fig 2-155 (11 Of 24)
Figure 2-155 (7 Of 24)
C

Figure 2-155 (5 Of 24) - Main Control Routine

Figure 2-155 (6 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the processing option is REPLACE, set the replace flag in the work area, go to the DELETE array routine, and go to the ADD array routine.		DPPXDBAS
2	If the processing option is TEST, set the test flag in the work area, go to the DELETE array routine, and go to the ADD array routine.		DPPXDBAS

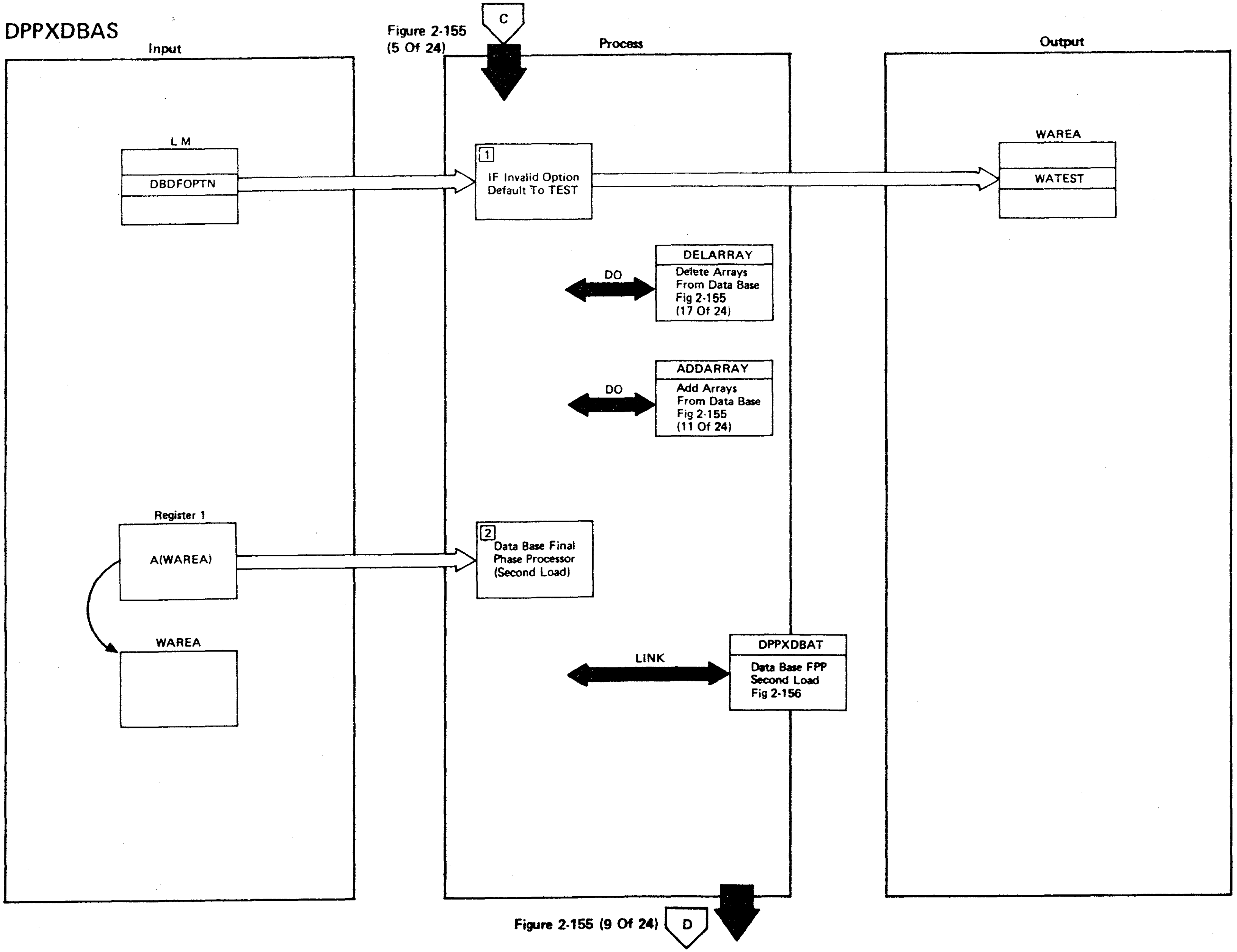


Figure 2-155 (7 Of 24) - Main Control Routine

Figure 2-155 (8 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the processing option is invalid, the test mode flag is set in the work area, and the message, INVALID OPTION-TEST ASSUMED, is written to SYSPRINT. DELETE array and ADD array processing is performed.	DPPXDB04I	DPPXDBAS
2	The second load of the data base final phase processor is entered via a LINK SVC to complete data base processing.		DPPXDBAS

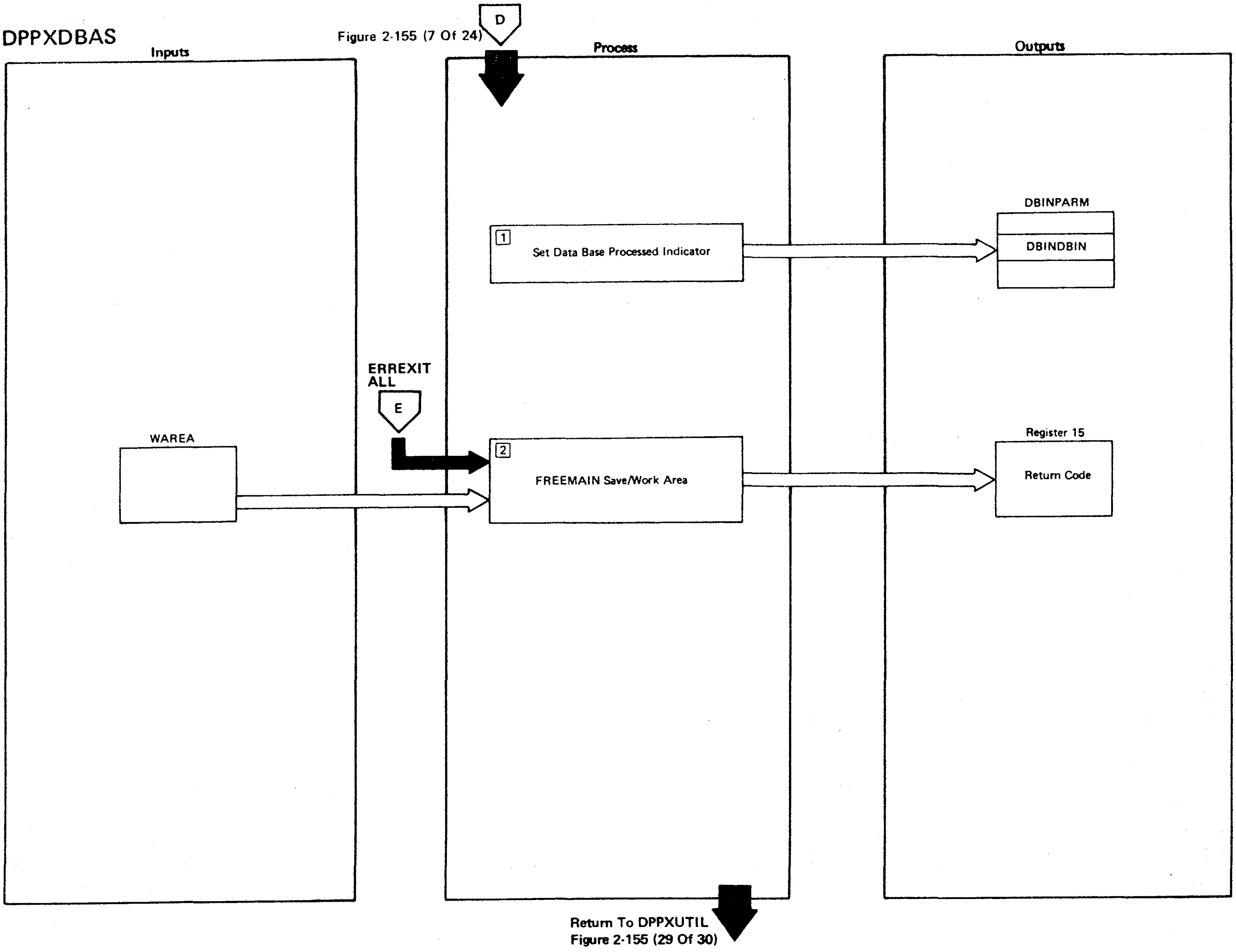


Figure 2-155 (9 Of 24) - Main Control Routine

Figure 2-155 (10 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Set data base processed indicator in the input parameters to tell DPPXUTIL that data base processing was performed.		DPPXDBAS
2	The DATA BASE FINAL PHASE PROCESSOR COMPLETED message is written to SYSPRINT, the save work area is freed, and the return code is set in register 15.	DPPXDB03I	DPPXDBAS

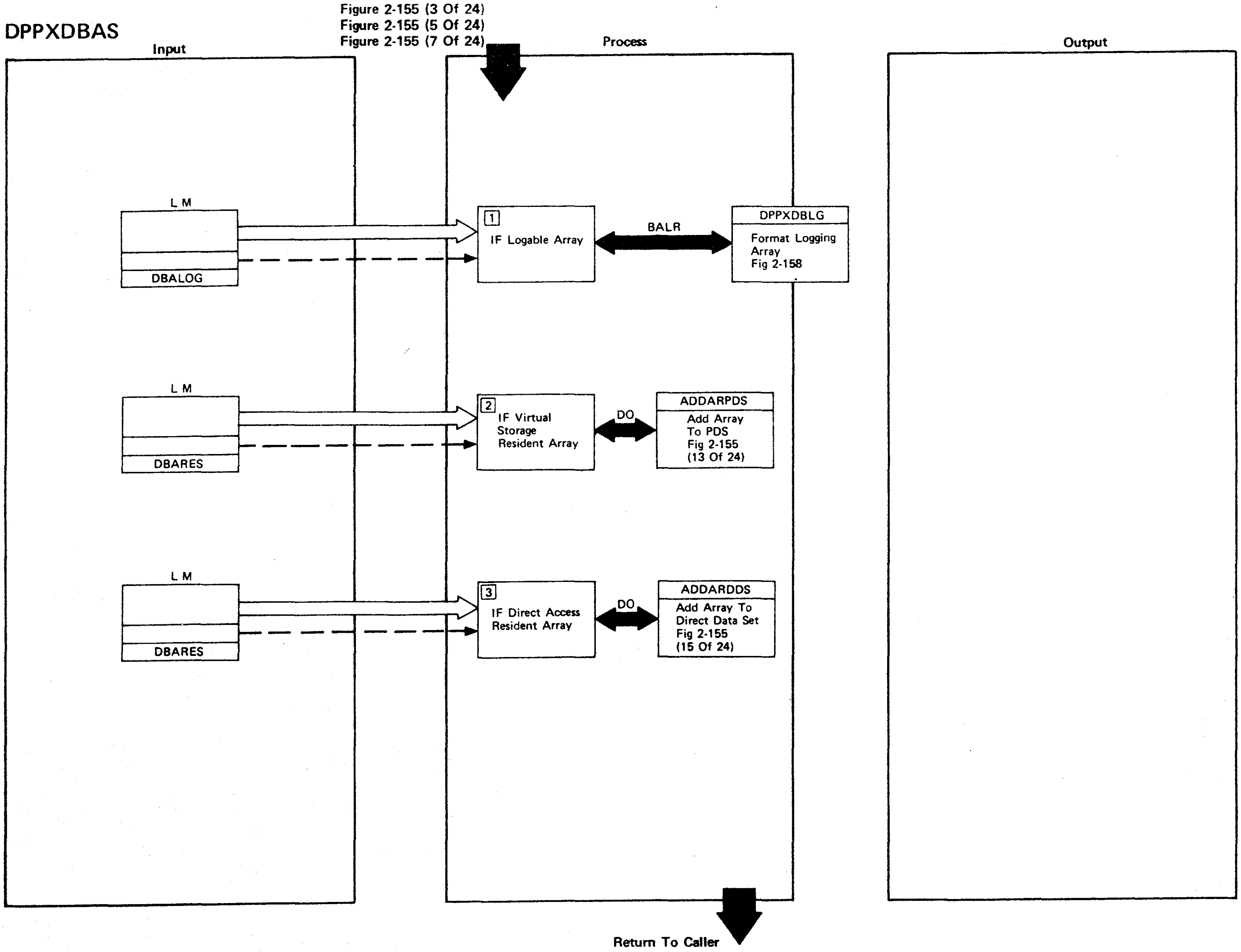


Figure 2-155 (11 Of 24) - ADDARRAY - Add Arrays To Data Base

Figure 2-155 (12 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the array being processed is a loggable virtual storage resident array, BALR to the logging array formatter.		DPPXDBAS
2	If the array is a virtual storage resident array, go to the routine to ADD arrays to the data base PDS.*		DPPXDBAS
3	If the array is a direct access resident array, go to the routine to ADD arrays to the data base direct data set.*		DPPXDBAS
	*These messages may be written to SYSPRINT:		
	DUPLICATE ARRAY NAME	DPPXDB06I	
	DUMMY BIT SET	DPPXDB17I	
	BLDL I/O ERROR MESSAGE	DPPXDB18I	

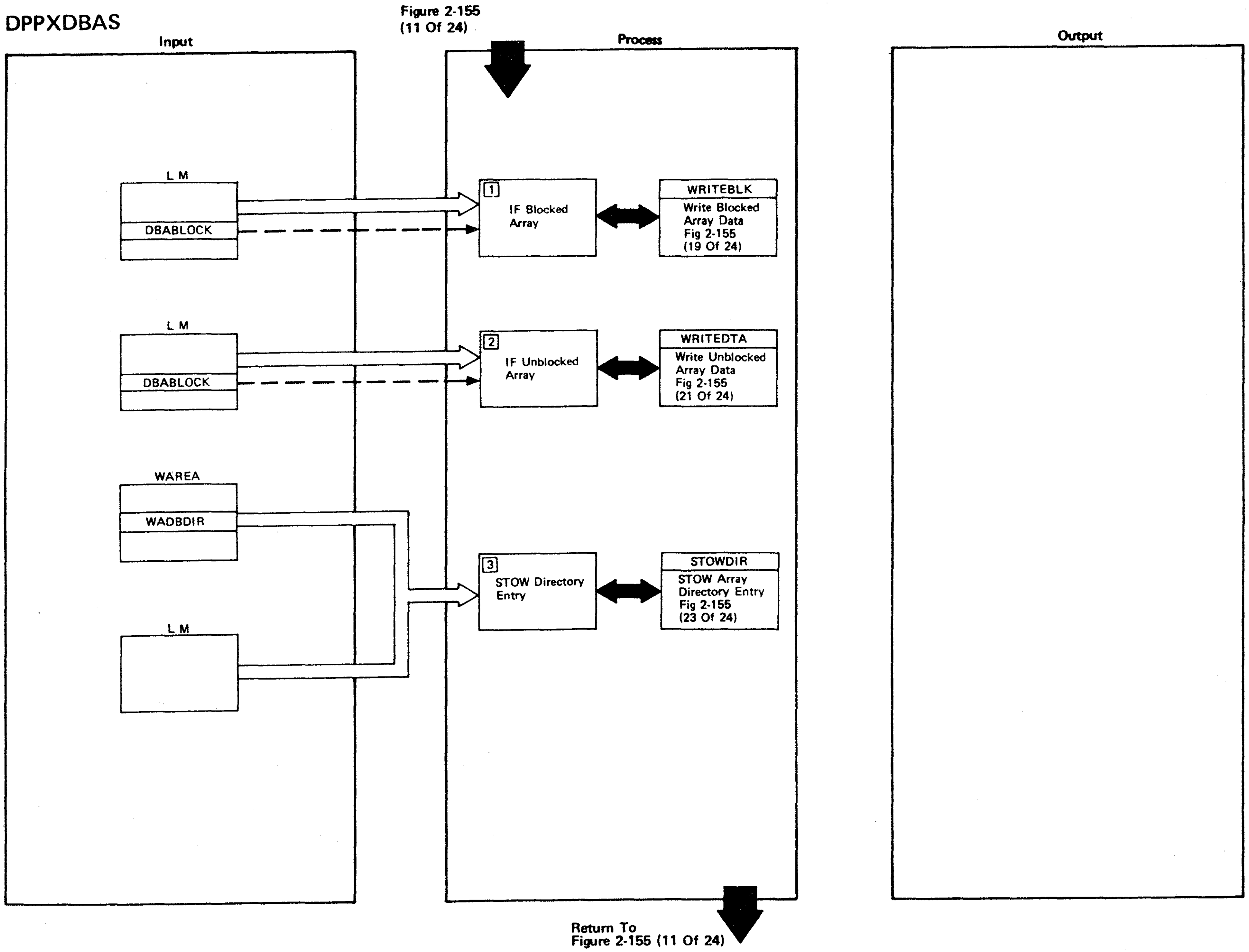


Figure 2-155 (13 Of 24) - ADDARPDS - Add Arrays To Data Base PDS

Figure 2-155 (14 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the array is blocked, go to the routine to write blocked array data to the data base PDS.		DPPXDBAS
2	If the array is unblocked, go to the routine to write unblocked array data to the data base PDS.		DPPXDBAS
3	STOW is used to write a dummy directory entry and an end-of-file at the end of the PDS array data. The real directory entry is moved to the new directory buffer.		DPPXDBAS

DPPXDBAS

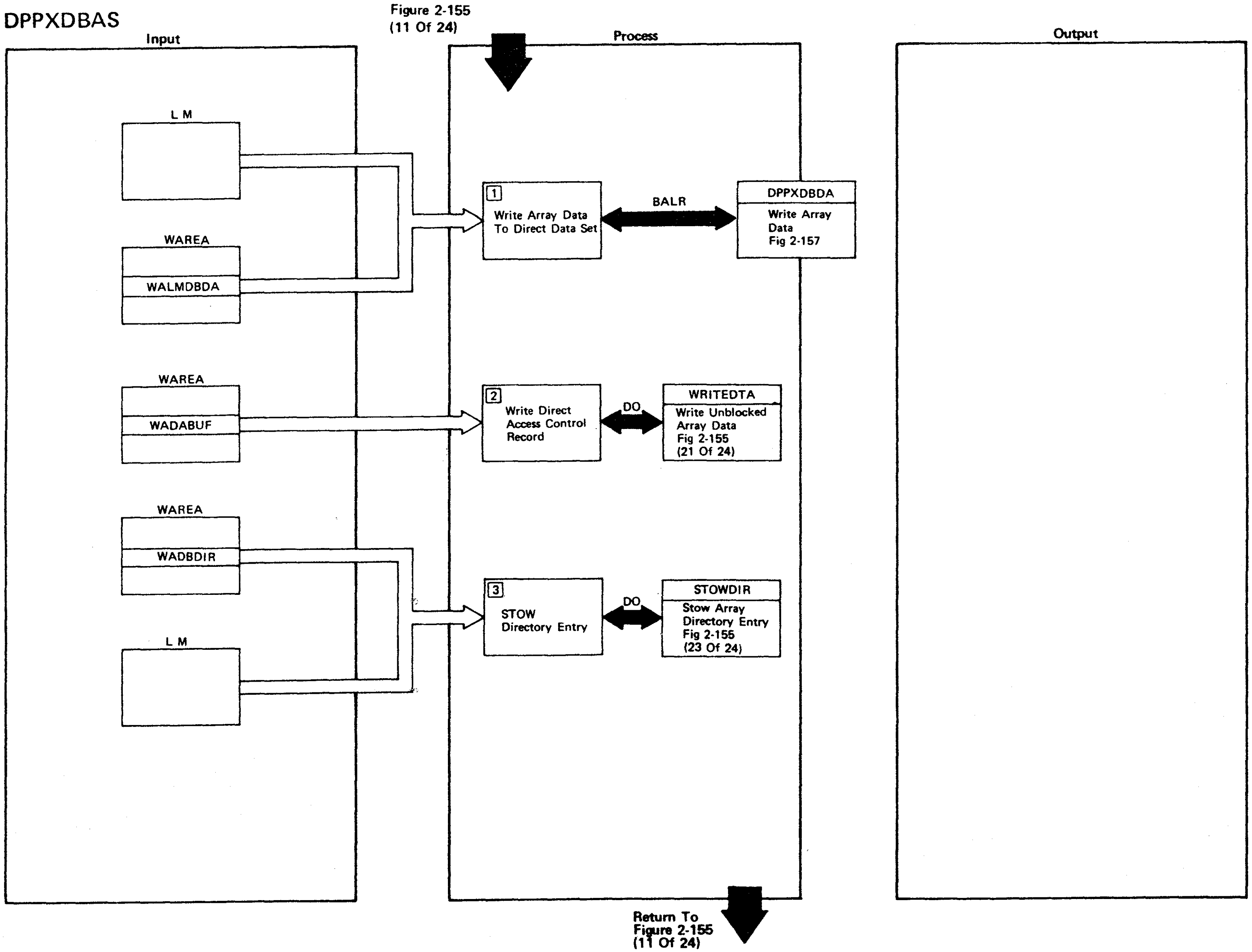


Figure 2-155 (15 Of 24) - ADDARDDS - Add Arrays To Data Base Direct Data Set

Figure 2-155 (16 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	BALR to DPPXDBDA to write array data to the data base direct data set.		DPPXDBAS
2	Go to the routine to write unblocked array data to write the direct access control record to the data base PDS.		DPPXDBAS
3	STOW is used to write a dummy directory entry and an end-of-file at the end of the PDS array data. The real directory entry is moved to the new directory buffer.		DPPXDBAS

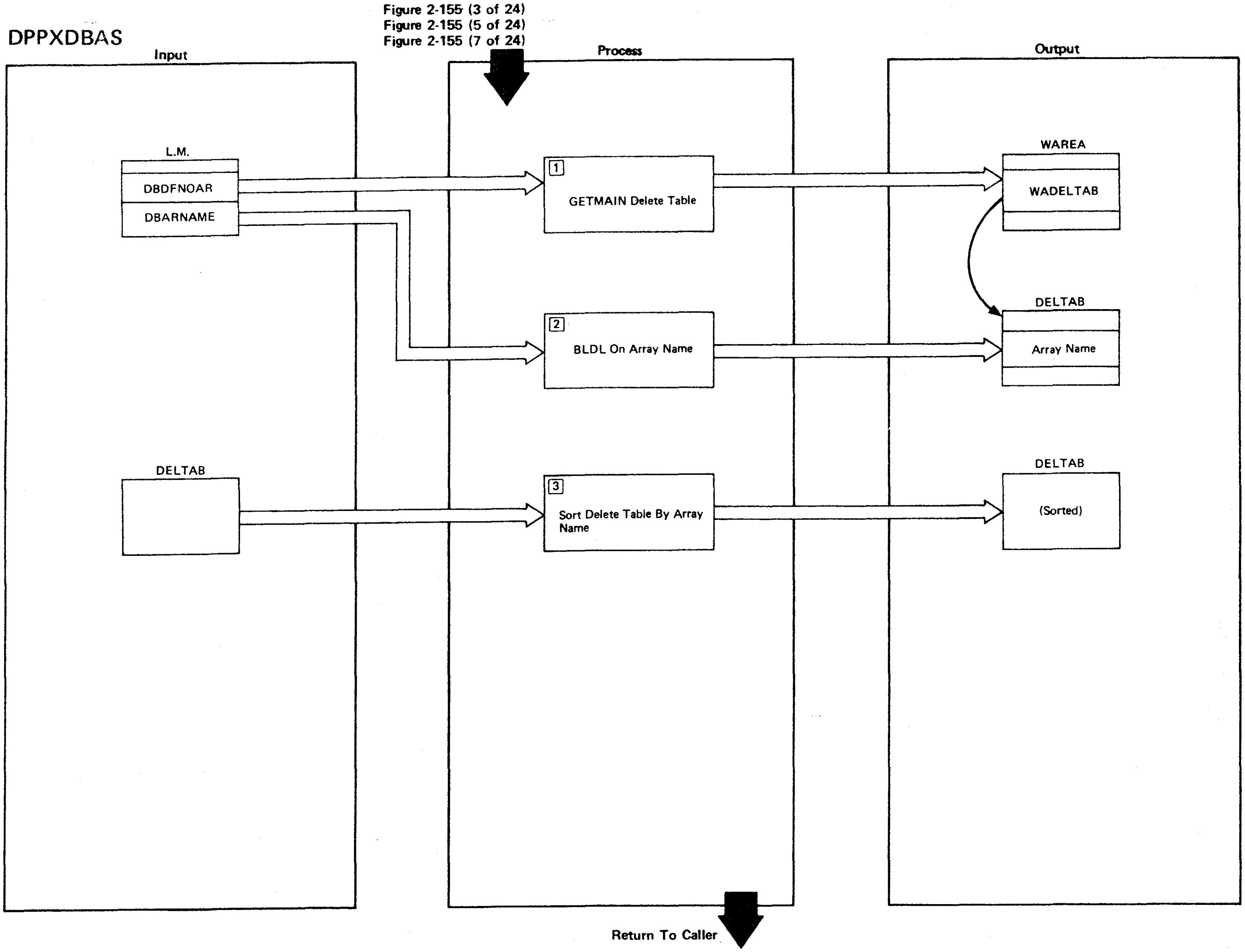


Figure 2-155 (17 Of 24) - DELARRAY - Delete Array From Data Base

Figure 2-155 (18 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	GETMAIN is used to obtain storage for the DELETE table.		DPPXDBAS
2	BLDL is used to determine if the array exists. If the array is found on the data base PDS directory, the array name is saved in the DELETE table.*		DPPXDBAS
3	The DELETE table is sorted in ascending sequence by array name.		DPPXDBAS
	*These messages may be written to SYSPRINT.		
	ARRAY DELETED	DPPXBD12I	
	ARRAY NOT FOUND	DPPXDB15I	
	DUMMY BIT SET	DPPXDB17I	
	BLDL I/O ERROR	DPPXDB18I	

DPPXDBAS

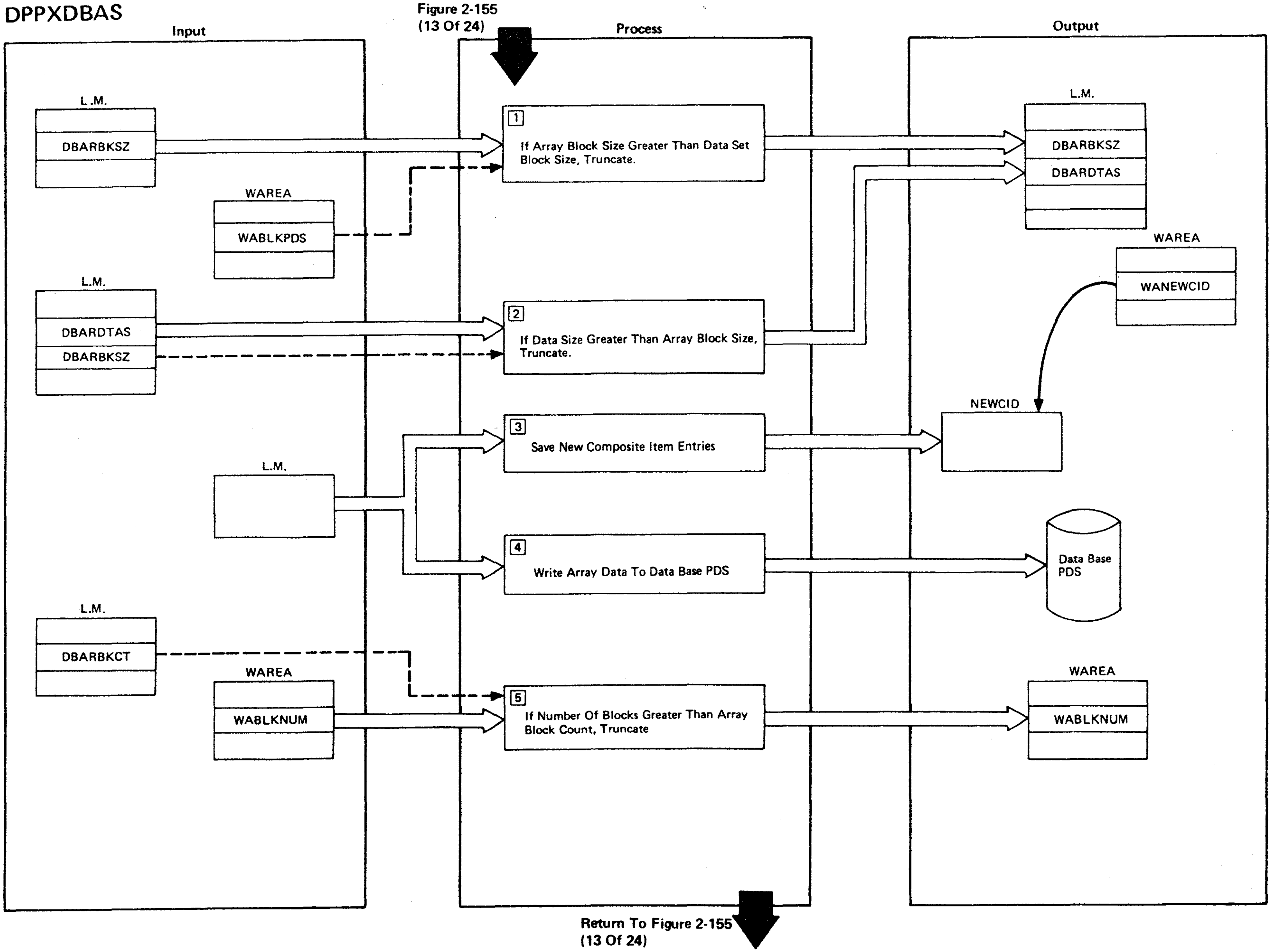


Figure 2-155
(13 Of 24)
Input
Process
Output
L.M.
DBARBKSZ
WAREA
WABLKPDS
L.M.
DBARDTAS
DBARBKSZ
L.M.
L.M.
DBARBKCT
WAREA
WABLKNUM
1 If Array Block Size Greater Than Data Set Block Size, Truncate.
2 If Data Size Greater Than Array Block Size, Truncate.
3 Save New Composite Item Entries
4 Write Array Data To Data Base PDS
5 If Number Of Blocks Greater Than Array Block Count, Truncate
L.M.
DBARBKSZ
DBARDTAS
WAREA
WANEWCID
NEWCID
Data Base PDS
WAREA
WABLKNUM
Return To Figure 2-155
(13 Of 24)

Figure 2-155 (19 Of 24) - WRITEBLK - Write Blocked Array Data To PDS

Figure 2-155 (20 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the array blocksize is greater than the data set blocksize, set the array blocksize equal to the data blocksize. Write the ARRAY BLOCKSIZE TRUNCATED message to SYSPRINT.	DPPXDB10I	DPPXDBAS
2	If the array data size is greater than the array blocksize, truncate the excess data and write the DATA TRUNCATED message to SYSPRINT.	DPPXDB09I	DPPXDBAS
3	Move the item control blocks for all named items into the new composite items buffer.		DPPXDBAS
4	Write blocked array data to the data base PDS. If assigned block numbers are out of sequence, default to the next sequential block number. Write the BLOCK NUMBER ERROR message to SYSPRINT.	DPPXDB20I	DPPXDBAS
5	If the number of blocks of array data written exceeds the array block count, truncate the excess blocks. Write the ARRAY BLOCK COUNT EXCEEDED message to SYSPRINT.	DPPXDB16I	DPPXDBAS

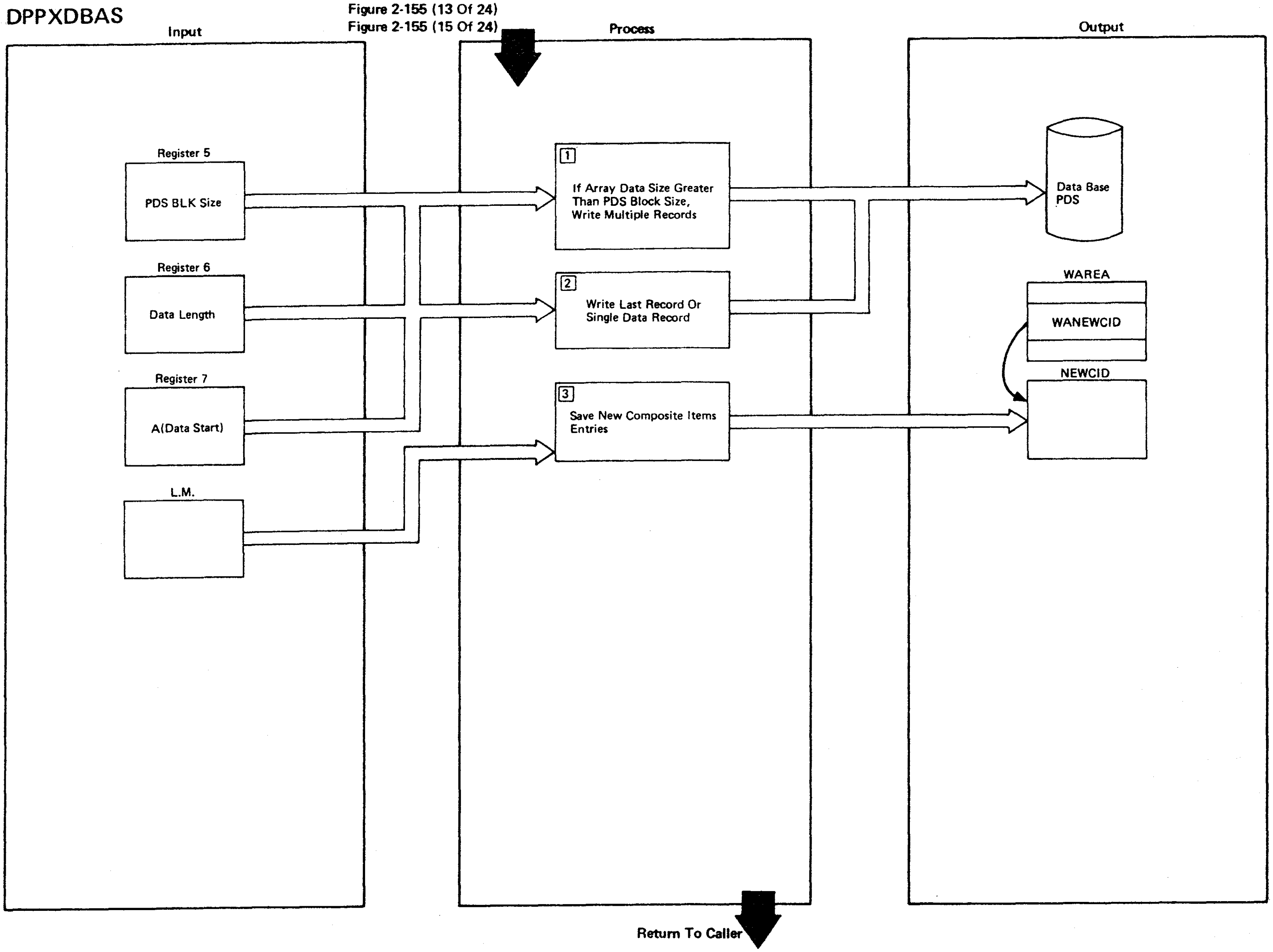


Figure 2-155 (21 Of 24) - WRITEDTA - Write Unblocked Array Data To PDS

Figure 2-155 (22 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the array data size is greater than the data base PDS blocksize, write PDS blocksize records until the size of the remaining data is less or equal to the PDS blocksize.		DPPXDBAS
2	Write one data record to the PDS which is less than or equal to the size of the data base PDS blocksize.		DPPXDBAS
3	Move the item control blocks for all named items into the new composite items buffer.		DPPXDBAS

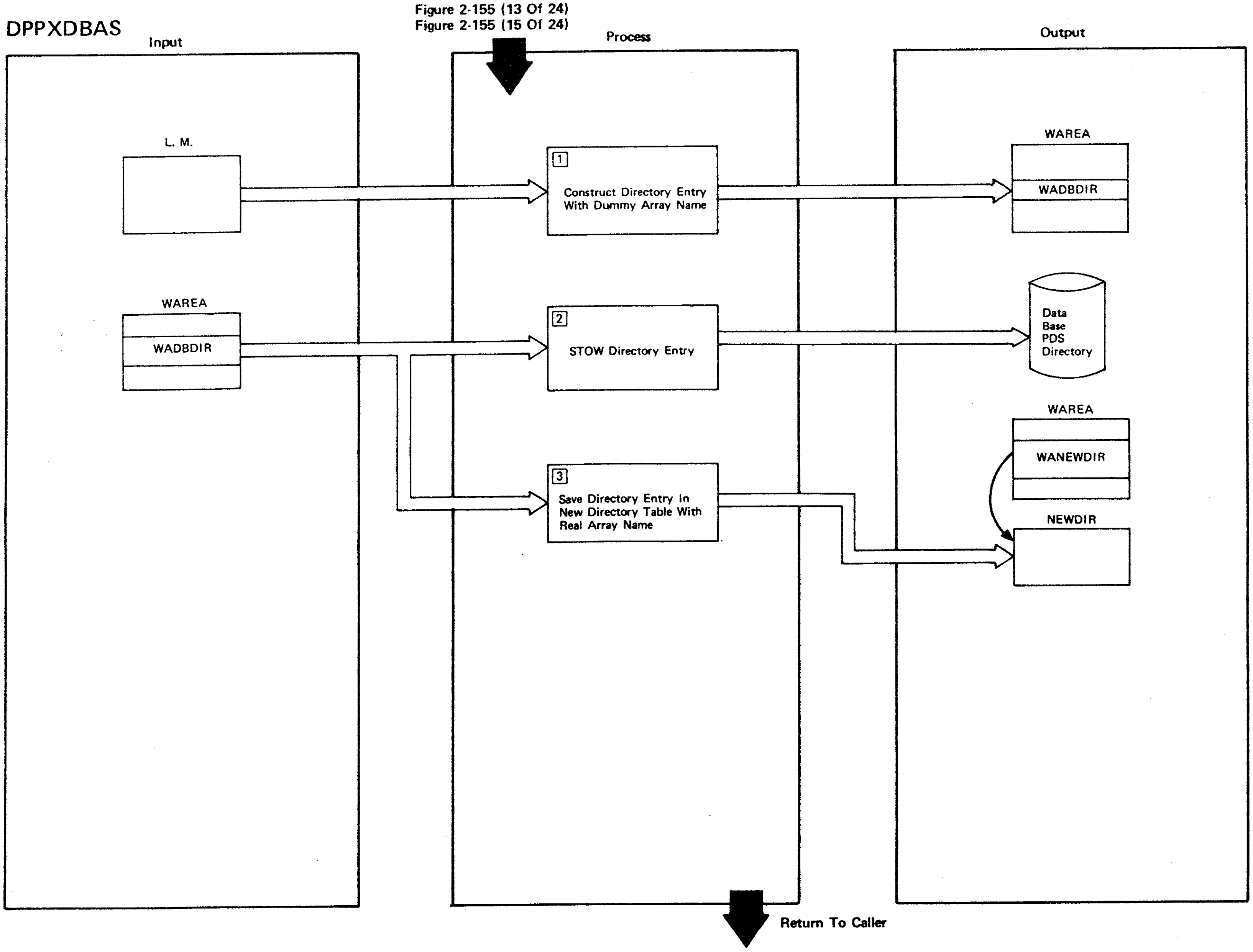


Figure 2-155 (23 Of 24) - STOWDIR - Stow PDS Directory Entry

Figure 2-155 (24 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Construct a new data bas- array directory entry with a dummy array name.*		DPPXDBAS
2	STOW the directory entry into the data base PDS directory. This will cause an end-of-file to be written at the end of the array data.		
3	The real array name is moved into the directory entry and the directory entry is saved in the new directory entry buffer.		
	*These messages may be written to SYSPRINT:		
	ARRAY ADDED	DPPXDB11I	
	ARRAY REPLACED	DPPXDB13I	
	ARRAY TESTED	DPPXDB14I	
	DUPLICATE ARRAY NAME	DPPXDB19I	

DPPXDBAT

Entered Via Link From DPPXDBAS
Figure 2-155 (7 Of 24)

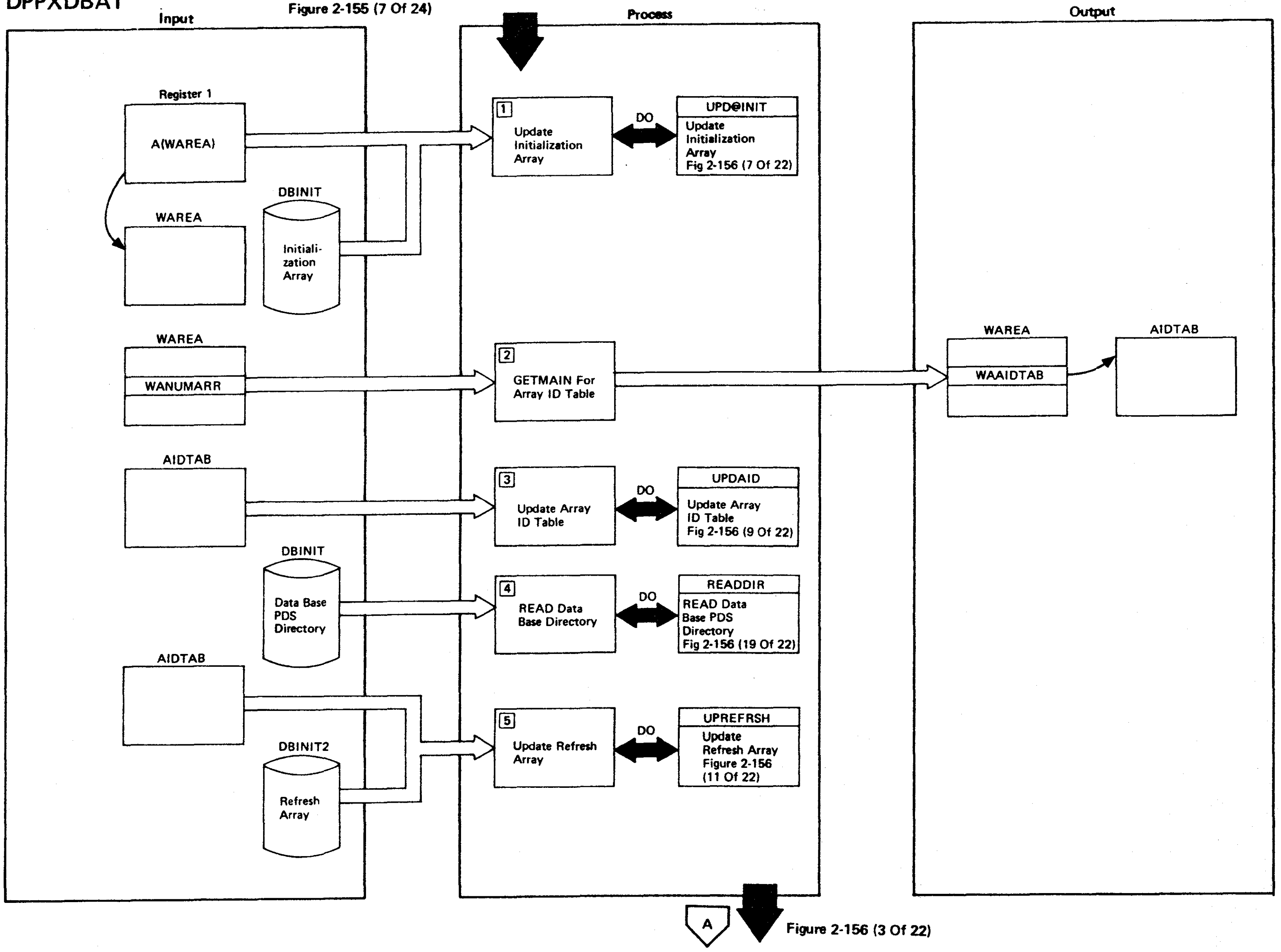


Figure 2-156 (1 Of 22) - Main Control Routine

Figure 2-156 (2 of 22).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Branch to the routine to update the initialization array.		DPPXDBAT
2	GETMAIN is used to obtain storage for the array ID table.		DPPXDBAT
3	Branch to the routine to update the array ID table.		DPPXDBAT
4	Branch to the routine to read the data base directory.		DPPXDBAT
5	Branch to the routine to update the refresh array, @REFRSH		DPPXDBAT

DPPXDBAT

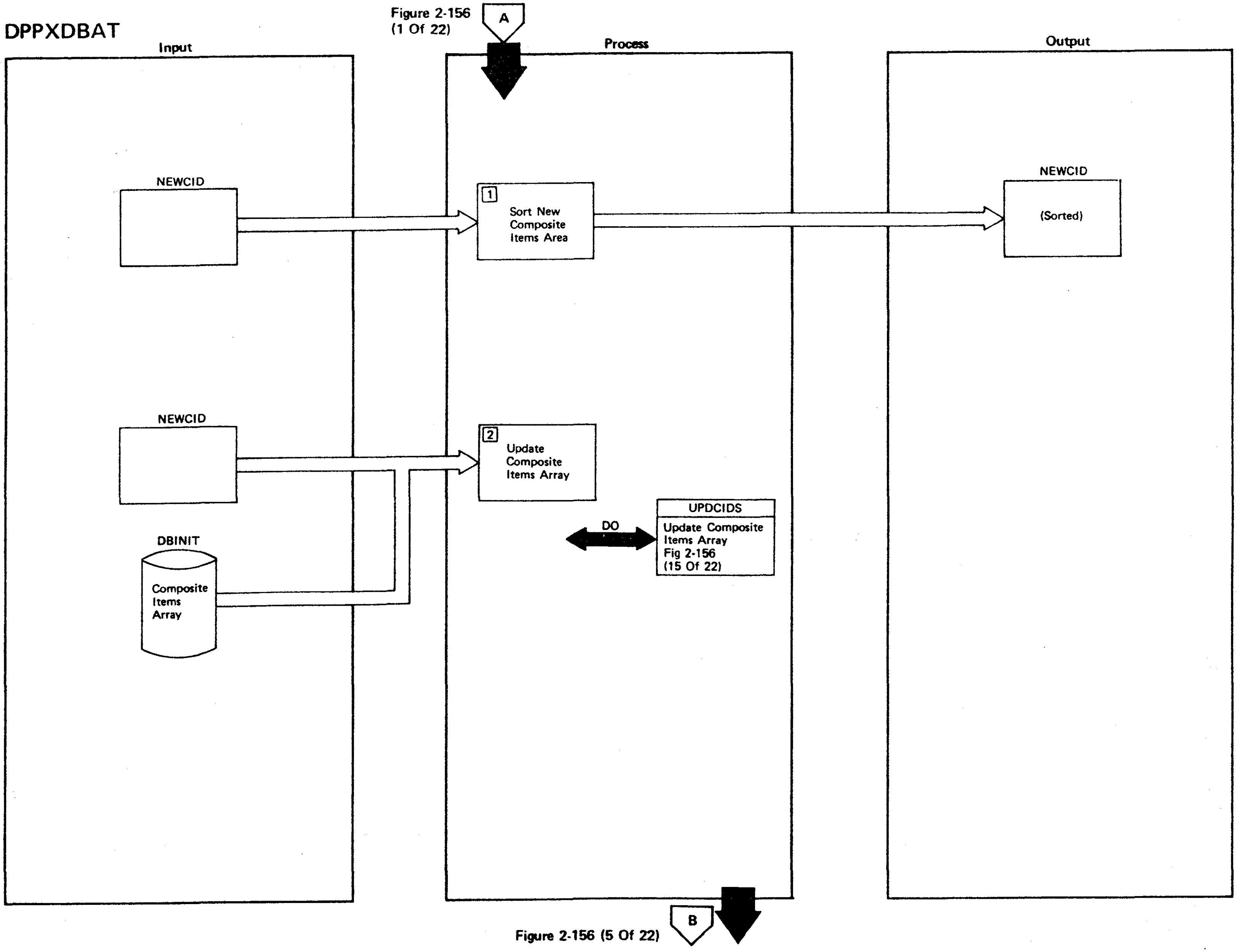


Figure 2-156 (3 Of 22) - Main Control Routine

Figure 2-156 (4 of 22).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Sort the new composite items table in ascending sequence by item name.		DPPXDBAT
2	Branch to the routine to update the composite items array.		DPPXDBAT

DPPXDBAT

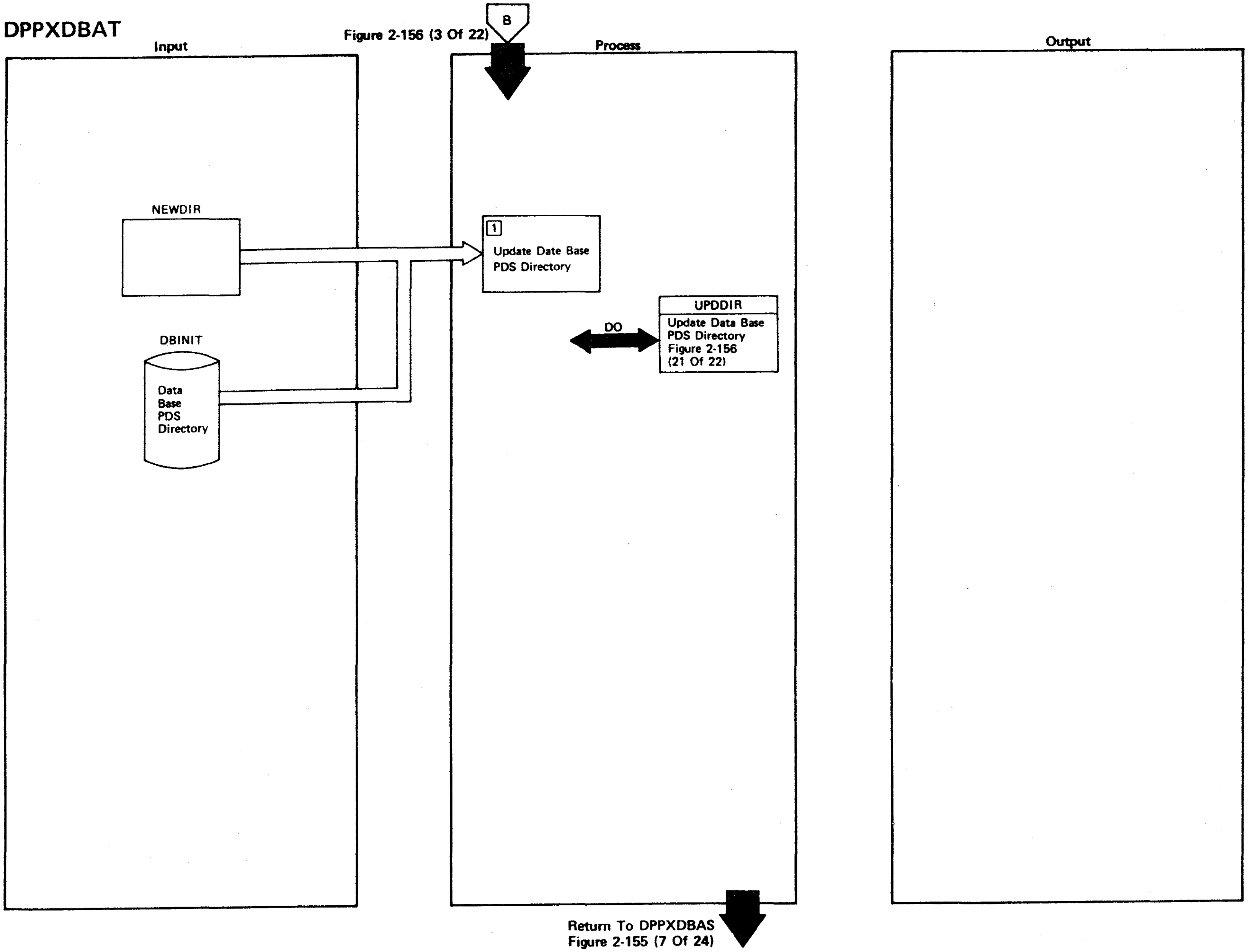

Input
Figure 2-156 (3 Of 22)
B
Process
Output
NEWDIR
DBINIT
Data Base PDS Directory
1
Update Date Base PDS Directory
DO
UPDDIR
Update Data Base PDS Directory Figure 2-156 (21 Of 22)
Return To DPPXDBAS Figure 2-155 (7 Of 24)

Figure 2-156 (5 Of 22) - Main Control Routine

Figure 2-156 (6 of 22).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Branch to the routine to update the data base PDS directory.		DPPXDBAT

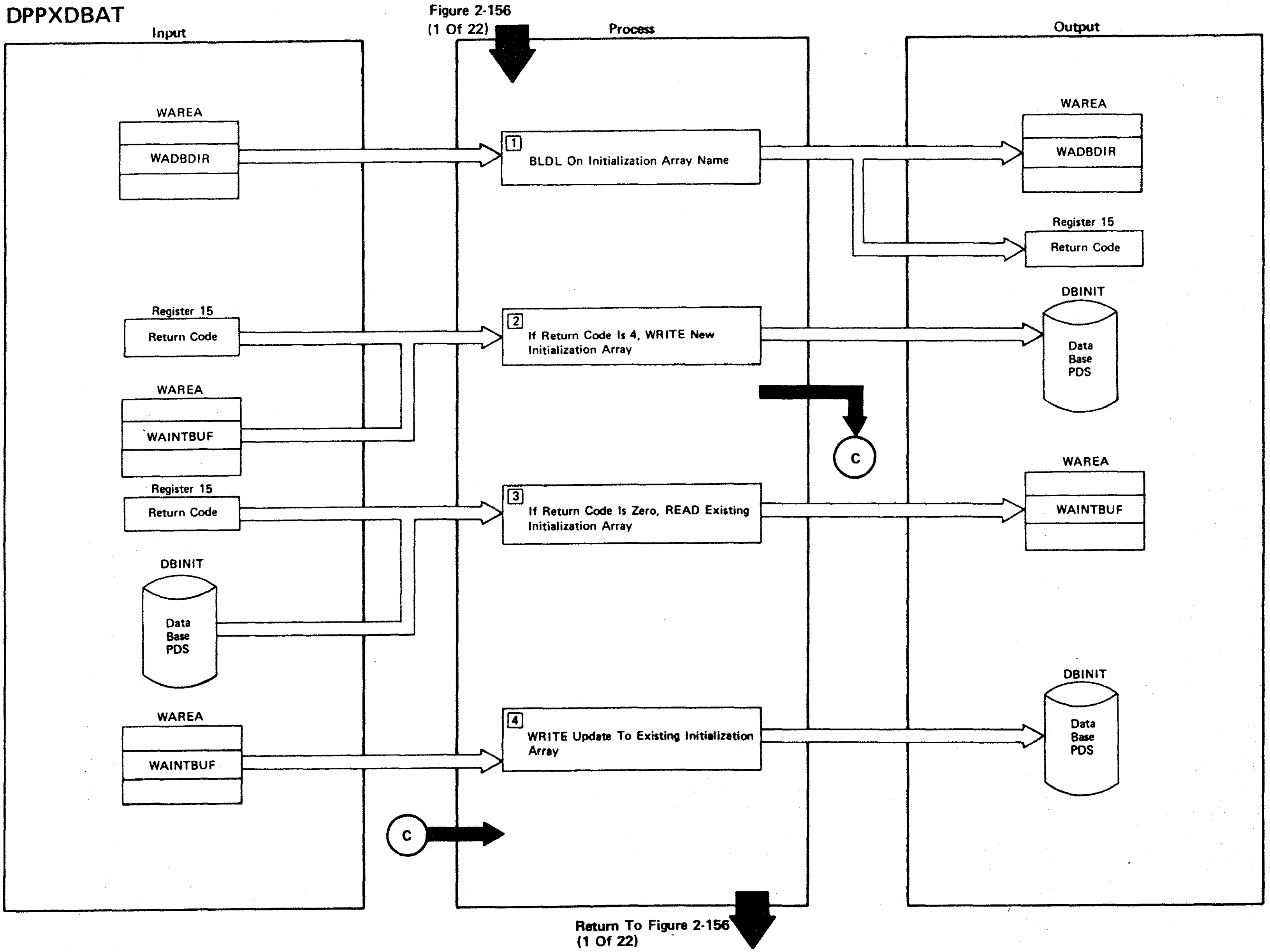


Figure 2-156 (7 Of 22) - UPD@INIT - Update Initialization Array

Figure 2-156 (8 of 22).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	BLDL is used to locate the data base initialization array on the data base PDS.		DPPXDBAT
2	If the BLDL return code is 4, no initialization array exists on the data base, and a new initialization array is written to the data base.		DPPXDBAT
3	If the BLDL return code is 0, the existing initialization array is read from the data base.		DPPXDBAT
4	The initialization array data is updated and written back to the data base PDS.		DPPXDBAT

DPPXDBAT

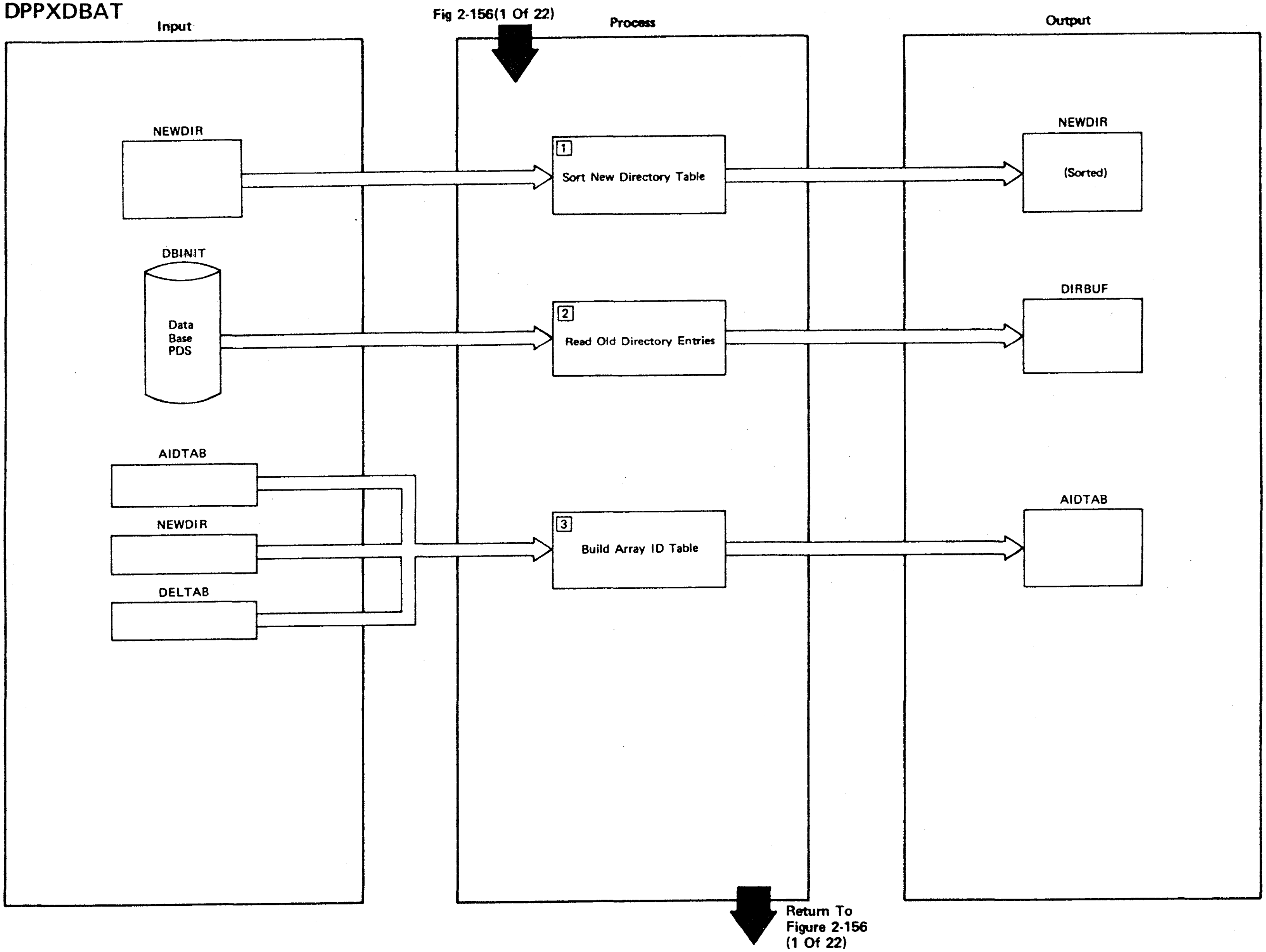


Figure 2-156 (9 Of 22) - UPDAID - Update Array ID Table

Figure 2-156 (10 of 22).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Sort the new directory table in ascending sequence by array name.		DPPXDBAT
2	The data base PDS directory is read from the data base data set and saved in the directory buffer.		DPPXDBAT
3	The contents of the array ID table are constructed.		DPPXDBAT

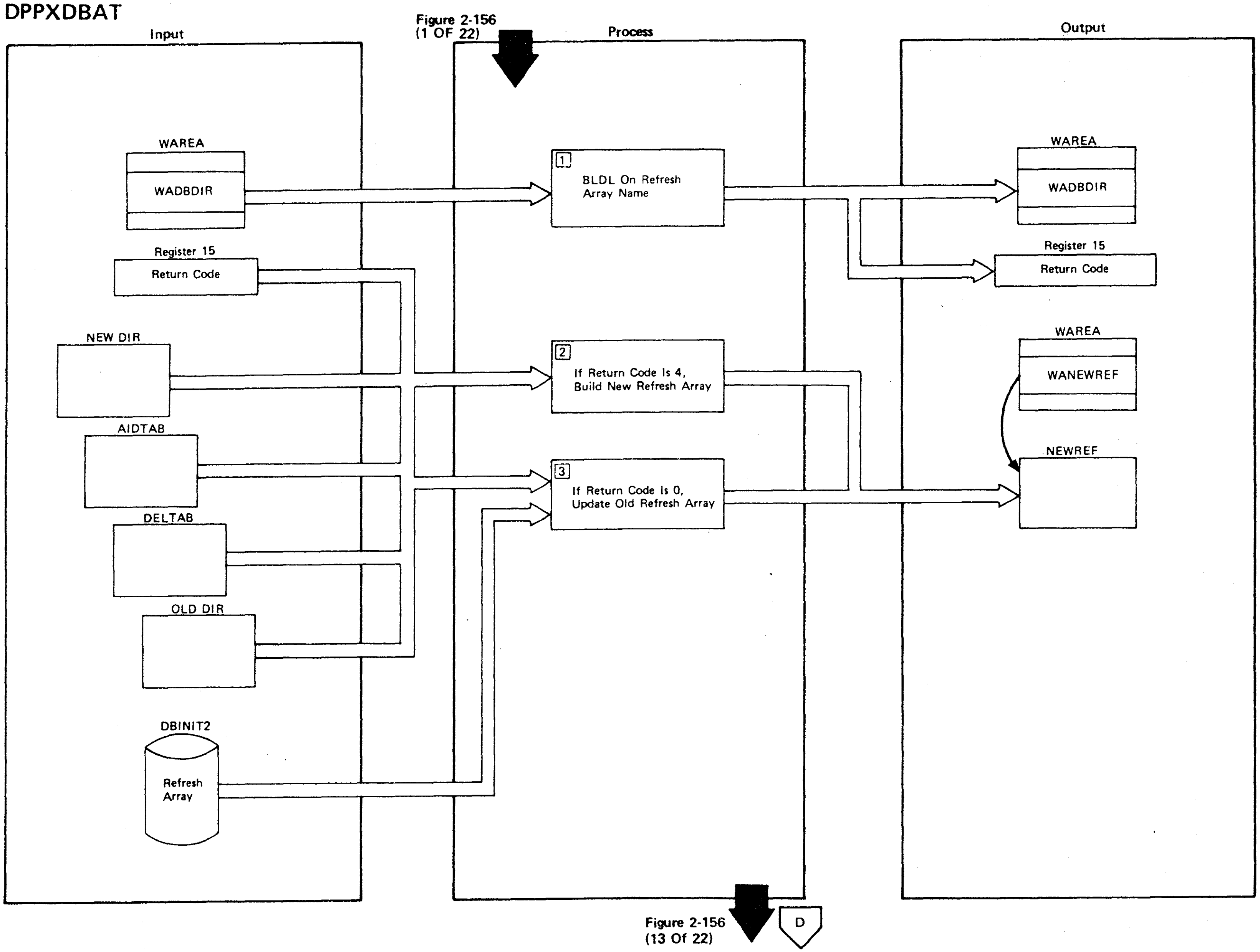


Figure 2-156 (11 Of 22) - UPREFRSH - Update Refresh Array

Figure 2-156 (12 of 22).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	BLDL is used to locate the refresh array on the data base PDS.		DPPXDBAT
2	If the BLDL return code is 4, the refresh array does not exist, and a new refresh array is constructed.		DPPXDBAT
3	If the BLDL return code is 0, the existing refresh array is read from the data base PDS and updated.		DPPXDBAT

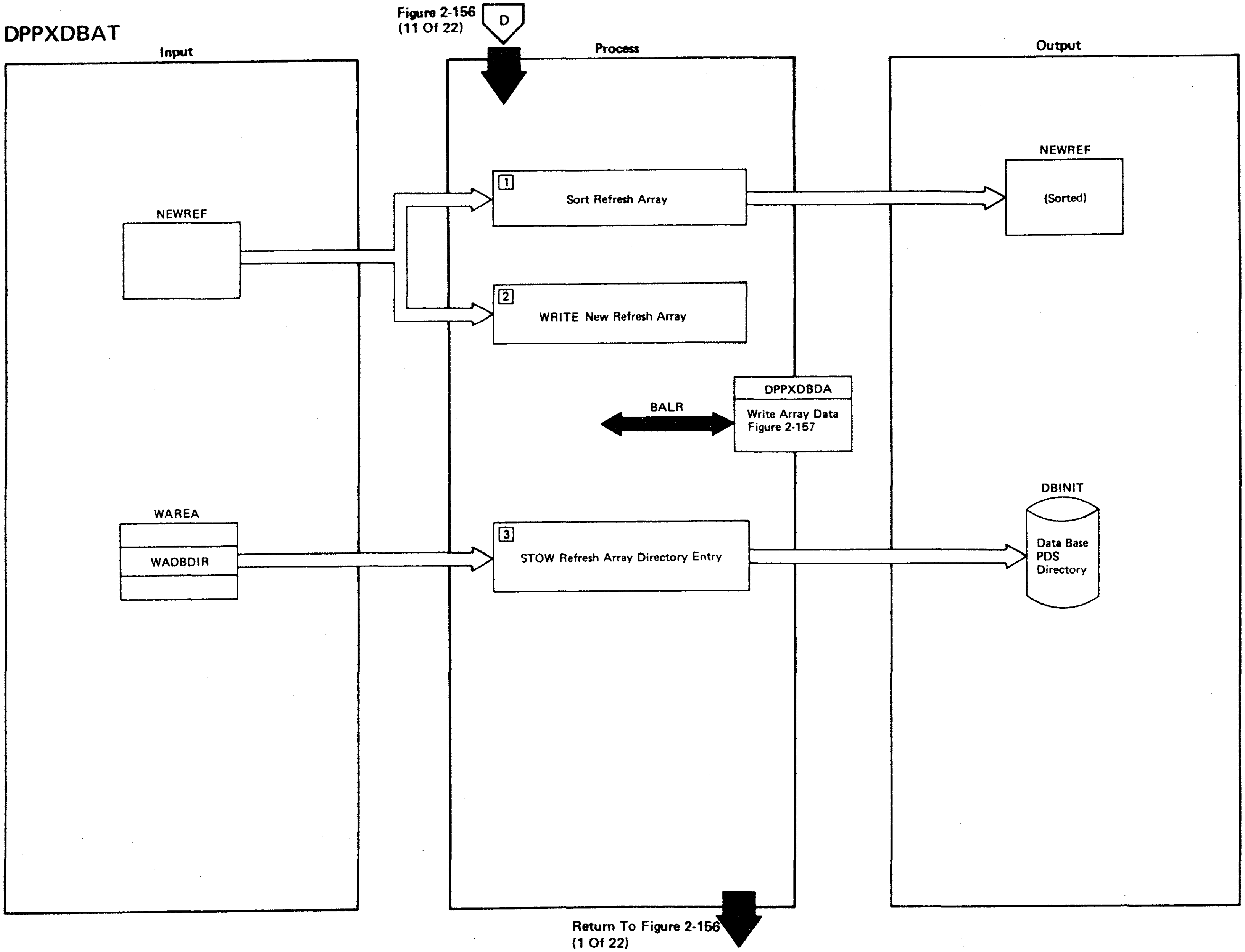


Figure 2-156 (13 Of 22) - UPREFRSH - Update Refresh Array

Figure 2-156 (14 of 22).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The refresh array is sorted in ascending sequence by array ID.		DPPXDBAT
2	BALR to DPPXDBDA to write the new refresh array to the data base direct access data set.		DPPXDBAT
3	STOW the refresh array directory entry to the data base PDS.		DPPXDBAT

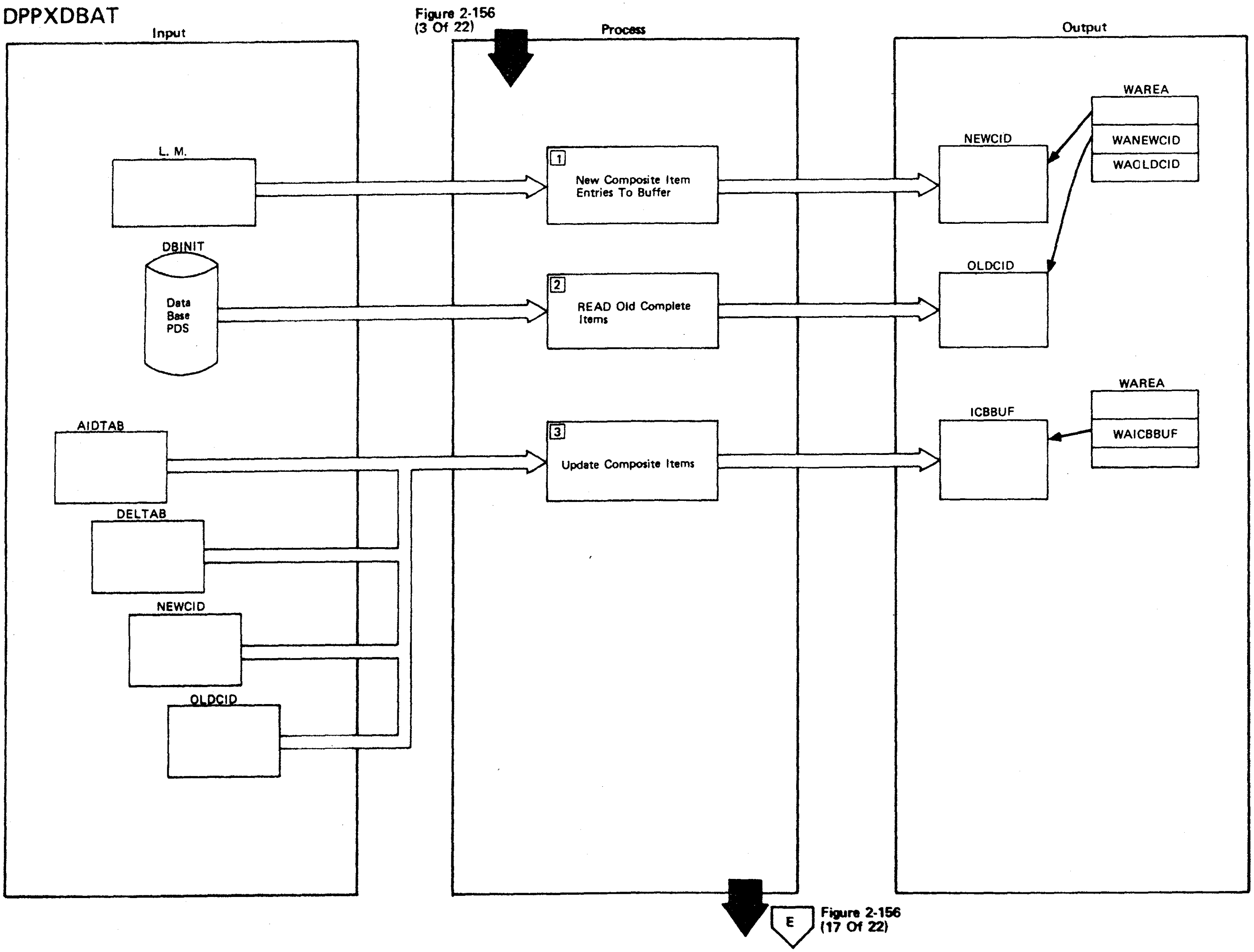


Figure 2-156 (15 Of 22) - UPDCIDS - Update Composite Items Array

Figure 2-156 (16 of 22).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Move new composite items to new composite items buffer.		DPPXDBAT
2	READ the existing composite items array from the data base PDS.		DPPXDBAT
3	Update the composite items. If any duplicate item names exist, WRITE the DUPLICATE ITEM NAME message to SYSPRINT.	DPPXDB08I	DPPXDBAT

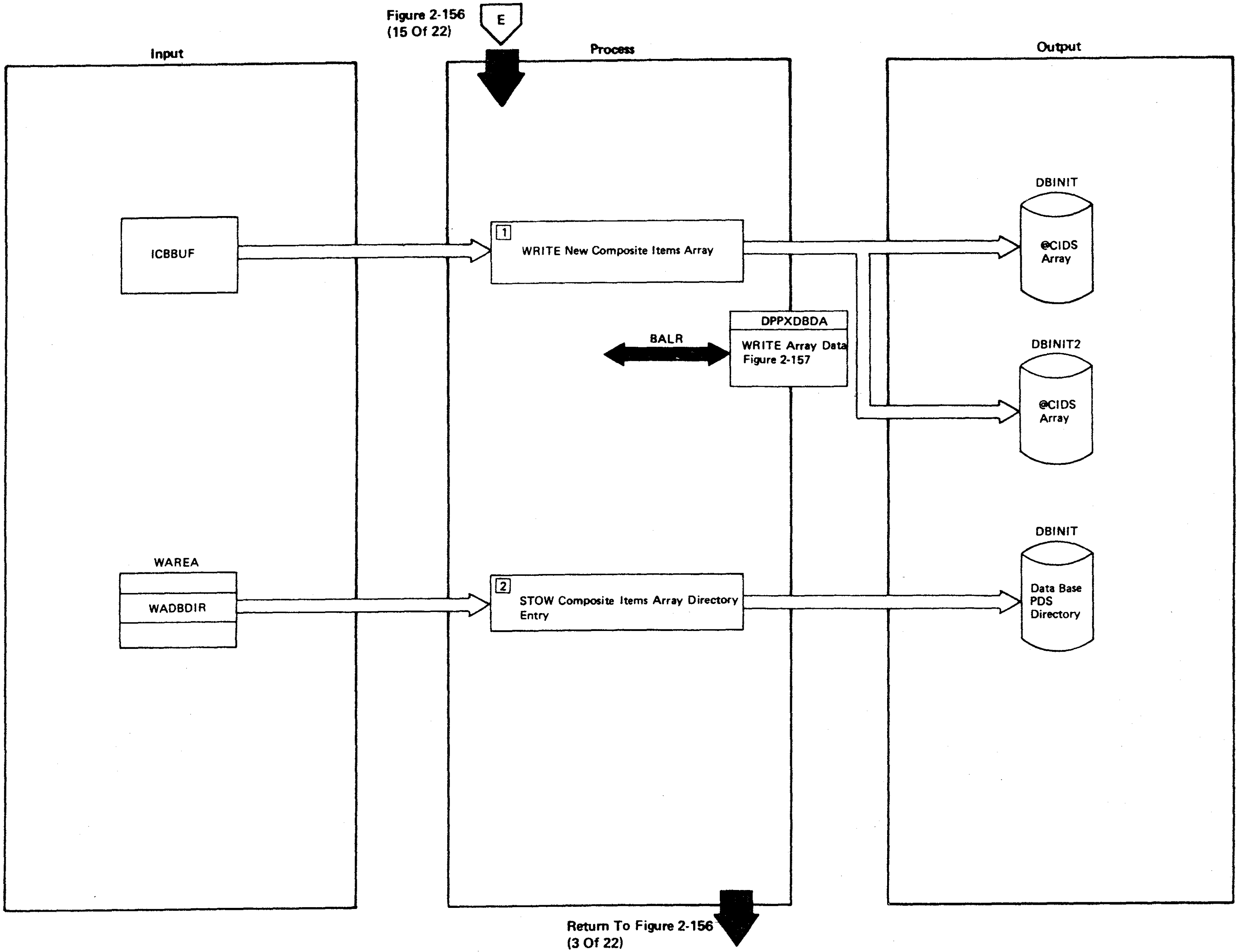


Figure 2-156 (17Of 22) - UPDCIDS - Update Composite Items Array

Figure 2-156 (18 of 22).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	WRITE the new composite items array to the data base PDS. BALR to DPPXDBDA to WRITE the composite items array to the data base direct access data set.		DPPXDBAT
2	STOW the composite items array directory entry to the data base PDS directory entry.		DPPXDBAT

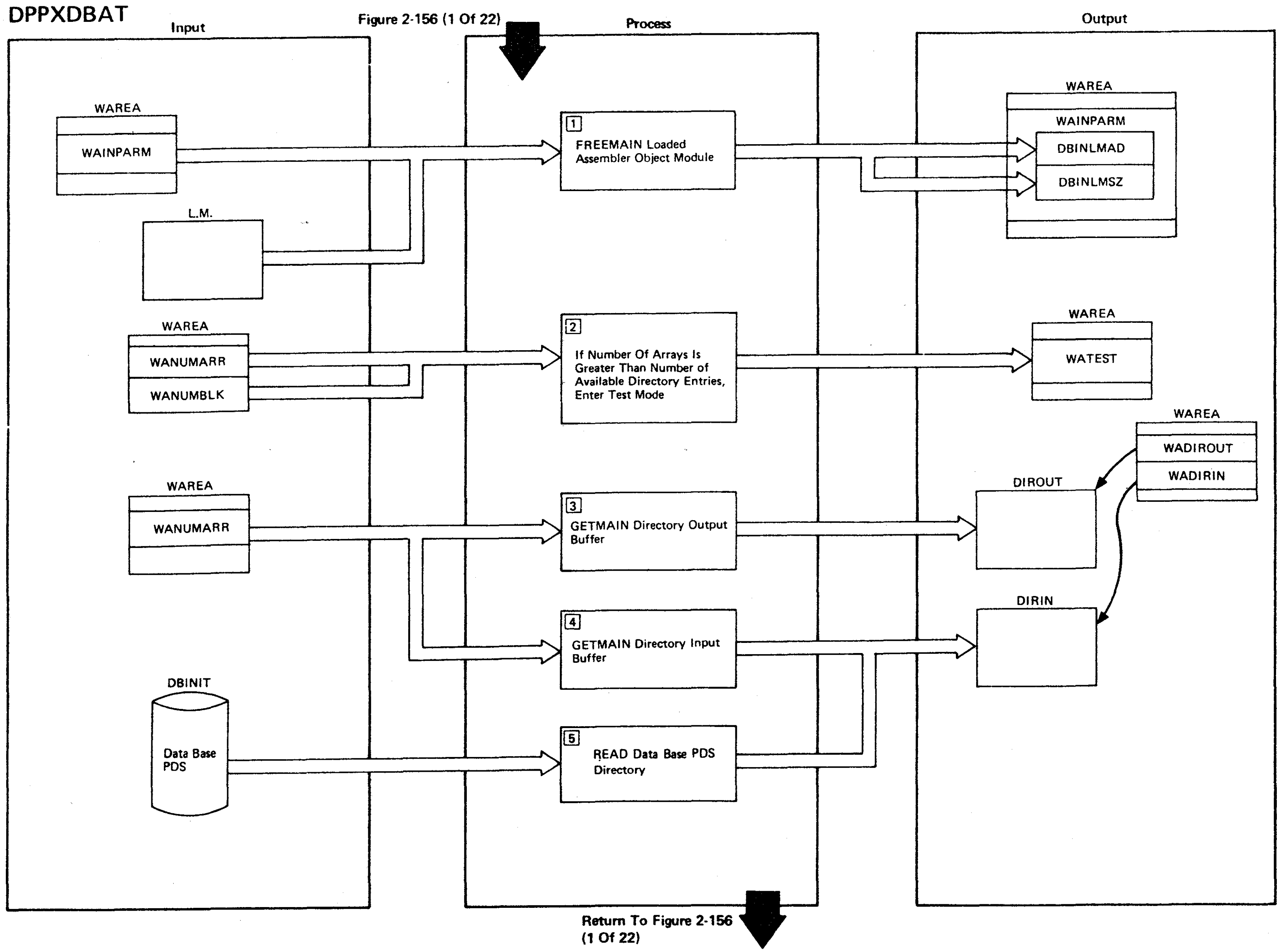

DPPXDBAT
Input
Figure 2-156 (1 Of 22)
Process
Output
WAREA
WAINPARM
L.M.
WAREA
WANUMARR
WANUMBLK
WAREA
WANUMARR
DBINIT
Data Base PDS
1 FREEMAIN Loaded Assembler Object Module
2 If Number Of Arrays Is Greater Than Number of Available Directory Entries, Enter Test Mode
3 GETMAIN Directory Output Buffer
4 GETMAIN Directory Input Buffer
5 READ Data Base PDS Directory
WAREA
WAINPARM
DBINLMAD
DBINLMSZ
WAREA
WATEST
WAREA
WADIROUT
WADIRIN
DIROUT
DIRIN
Return To Figure 2-156 (1 Of 22)

Figure 2-156 (19 Of 22) - READDIR - Read Data Base PDS Directory

Figure 2-156 (20 of 22).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	FREEMAIN the input load module storage area, zero the address, and zero the size in the input parameter list.		DPPXDBAT
2	If the number of arrays to be added to the data base exceeds the number of available data base PDS directory entries, enter test mode and WRITE the INSUFFICIENT DIRECTORY SPACE-ALLOCATED message to SYSPRINT.	DPPXDB02I	DPPXDBAT
3	GETMAIN is used to obtain storage space for the data base PDS directory output buffer.		DPPXDBAT
4	GETMAIN is used to obtain storage space for the data base PDS directory input buffer.		DPPXDBAT
5	READ the data base PDS directory from the data base PDS.		DPPXDBAT

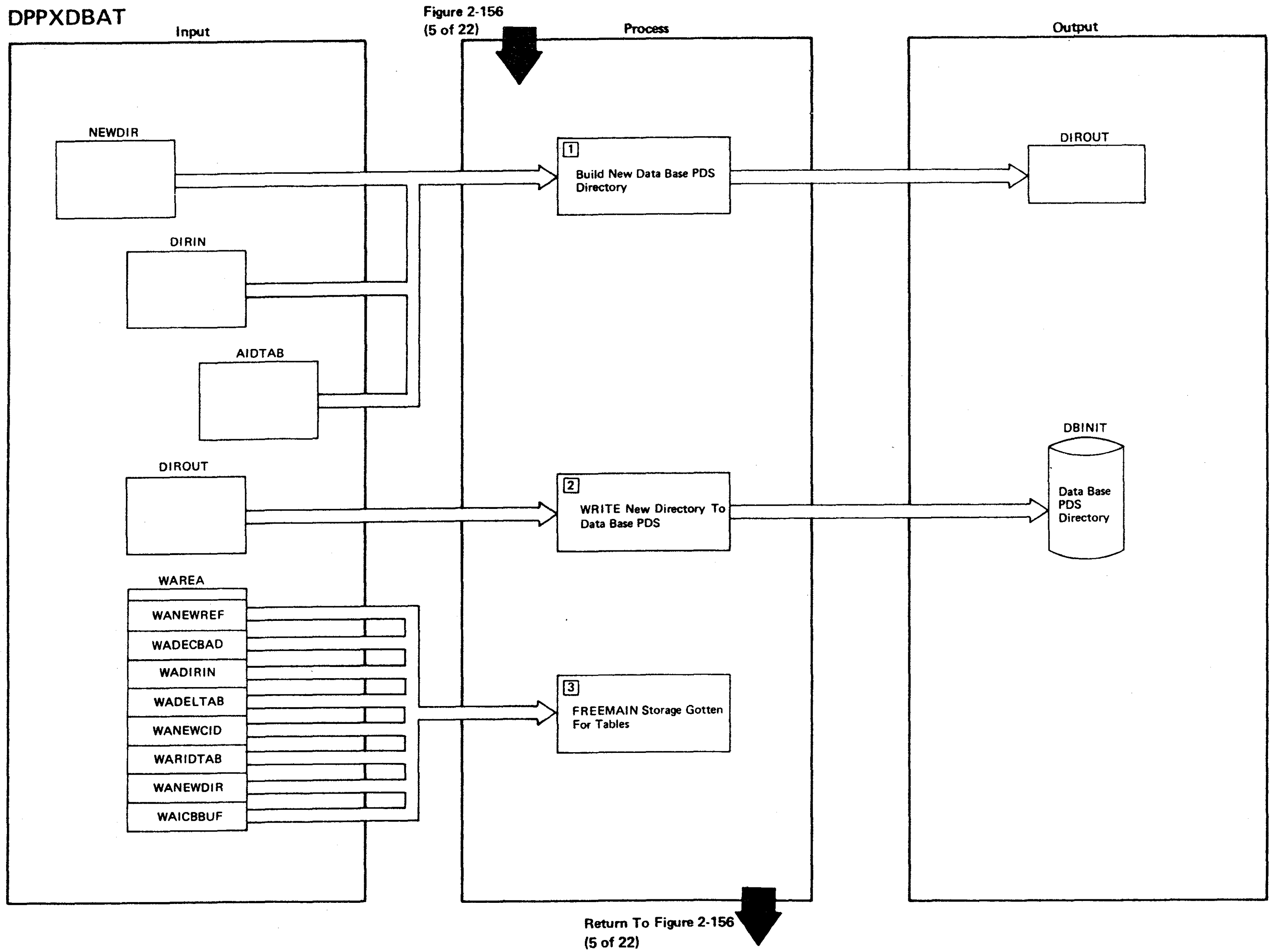


Figure 2-156 (21 Of 22) - UPDDIR - Update Data Base PDS Directory

Figure 2-156 (22 of 22).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Build a new data base PDS directory in the directory output buffer.		DPPXDBAT
2	WRITE the new data base PDS directory to the data base PDS.		DPPXDBAT
3	FREEMAIN all storage obtained for all tables that are no longer required.		DPPXDBAT

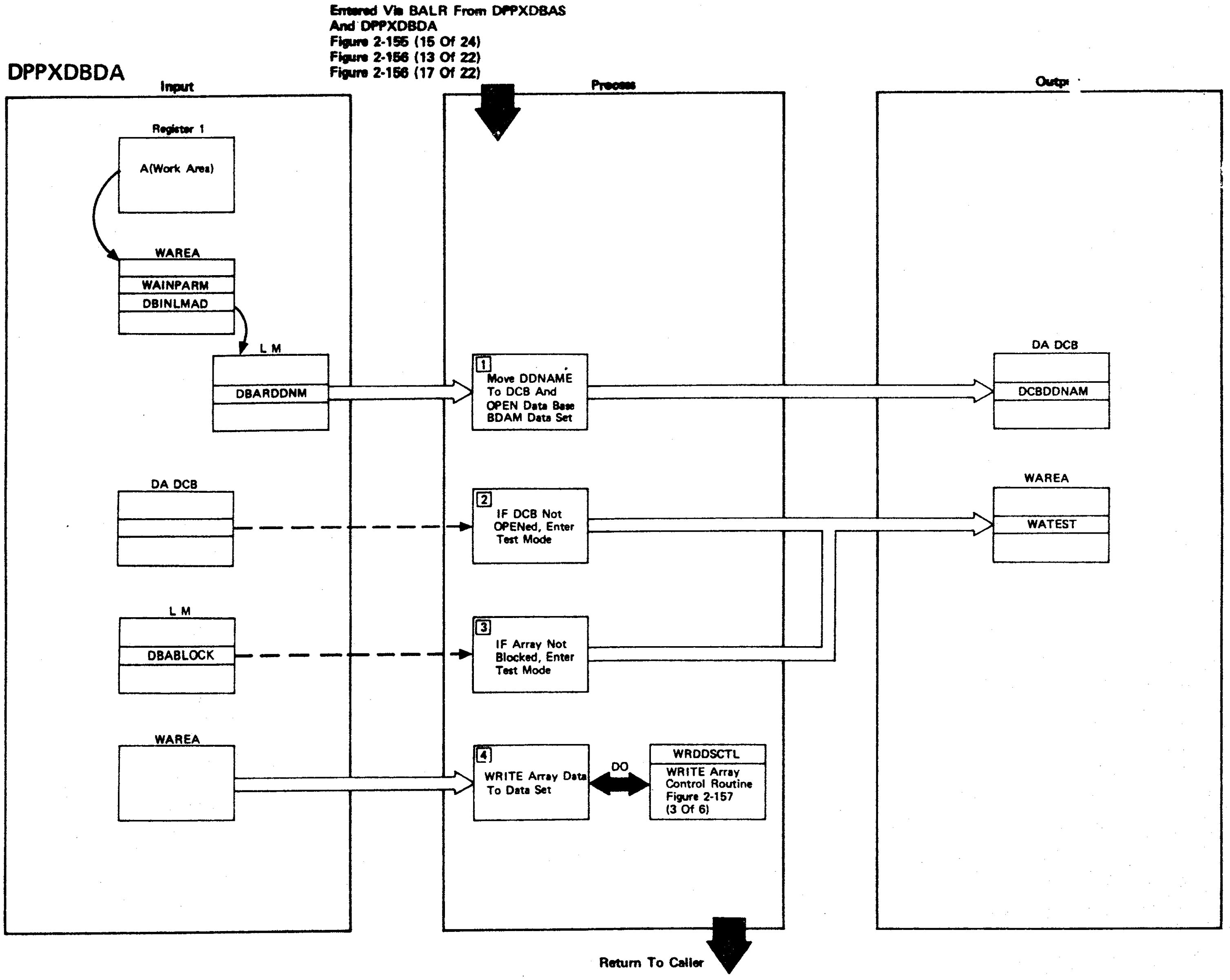

DPPXDBDA
Entered Via BALR From DPPXDBAS
And DPPXDBDA
Figure 2-155 (15 Of 24)
Figure 2-156 (13 Of 22)
Figure 2-156 (17 Of 22)
Input
Process
Outp
Register 1
A(Work Area)
WAREA
WAINPARM
DBINLMAD
L M
DBARDDNM
1 Move DDNAME To DCB And OPEN Data Base BDAM Data Set
DA DCB
DCBDDNAM
DA DCB
2 IF DCB Not OPENed, Enter Test Mode
WAREA
WATEST
L M
DBABLOCK
3 IF Array Not Blocked, Enter Test Mode
WAREA
4 WRITE Array Data To Data Set
DO
WRDDSCTL
WRITE Array Control Routine Figure 2-157 (3 Of 6)
Return To Caller

Figure 2-157 (1 Of 6) - Main Control Routine

Figure 2-157 (2 of 6).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Move the DDNAME of the direct access data set to the direct access DCB and OPEN the DA DCB.		DPPXDBDA
2	If the DCB does not open, enter test mode and write the UNABLE TO OPEN DDNAME message to SYSPRINT.	DPPXDB50I	DPPXDBDA
3	If the direct access array is not blocked, enter test mode and WRITE the message, NOT A BLOCKED ARRAY, to SYSPRINT.	DPPXDB5I	DPPXDBDA
4	WRITE array data to the direct access data set.		DPPXDBDA

DPPXDBDA

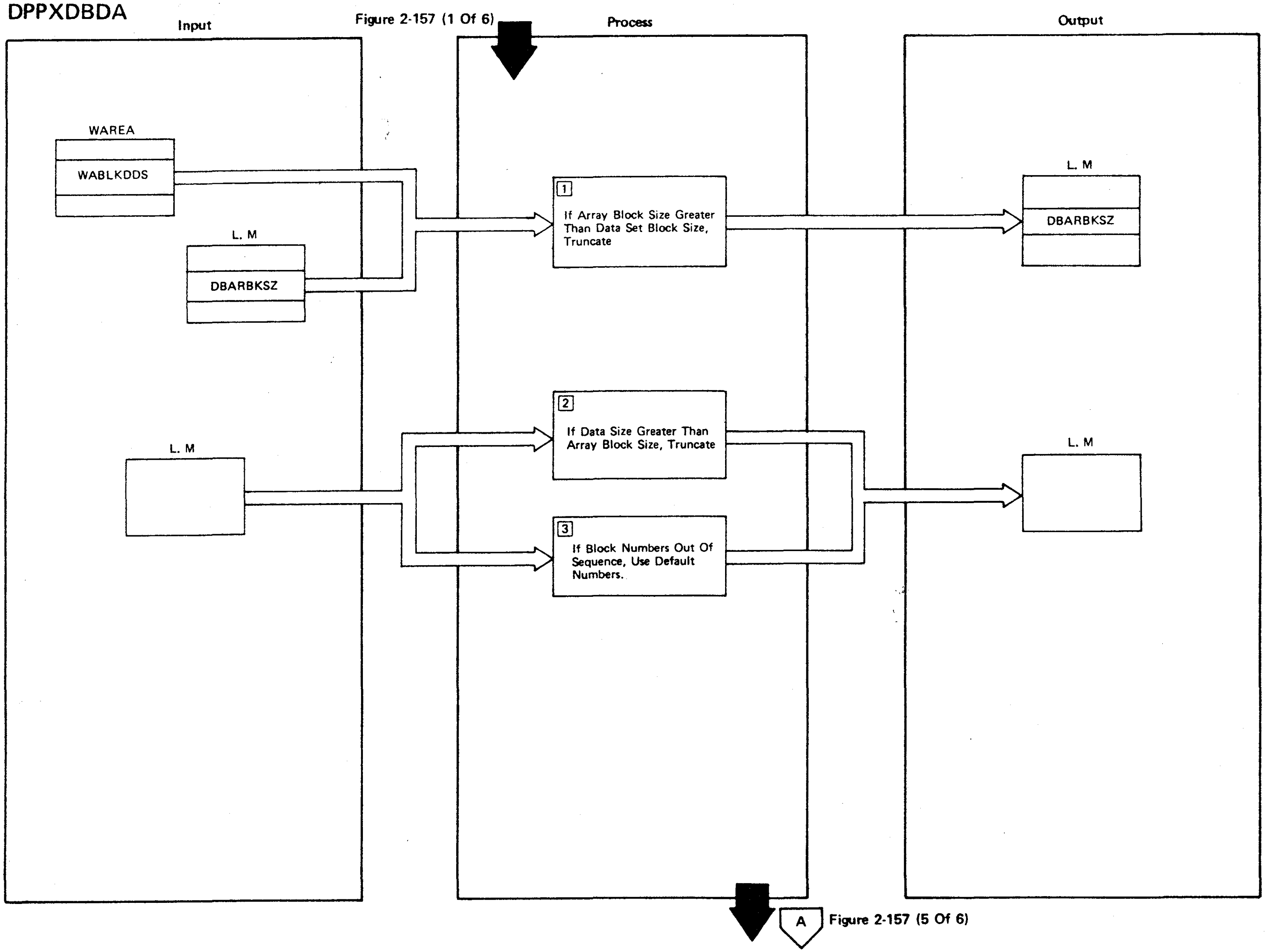

Input
Figure 2-157 (1 Of 6)
Process
Output
WAREA
WABLKDDS
L. M
DBARBKSZ
L. M
1
If Array Block Size Greater Than Data Set Block Size, Truncate
2
If Data Size Greater Than Array Block Size, Truncate
3
If Block Numbers Out Of Sequence, Use Default Numbers.
L. M
DBARBKSZ
L. M
A
Figure 2-157 (5 Of 6)

Figure 2-157 (3 Of 6) - WRDDSCTL - Write Array Data Control Routine

Figure 2-157 (4 of 6).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If the array blocksize is greater than the data set blocksize, use the data set blocksize as the array blocksize. The message, ARRAY BLOCKSIZE TRUNCATED, is written to SYSPRINT.	DPPXDB52I	
2	If the data size is greater than the array blocksize, truncate the data to the array blocksize. The message, DATA TRUNCATED, is written to SYSPRINT.	DPPXDB53I	
3	If a block number is defined out of sequence, default the block number to the next sequential block number. The message, BLOCK NUMBER ERROR, is written to SYSPRINT.	DPPXDB55I	

DPPXDBDA

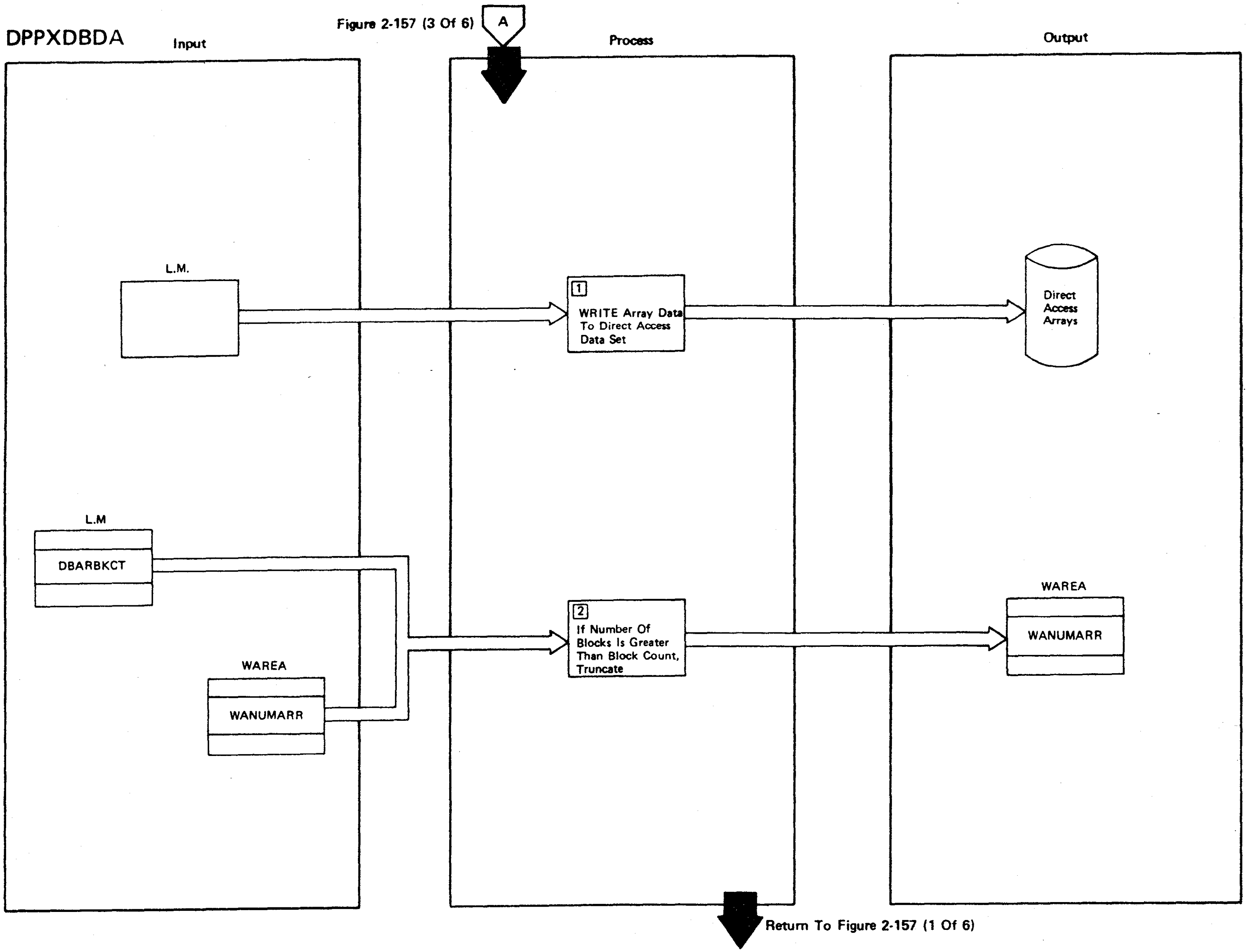

Input
Figure 2-157 (3 Of 6)
A
Process
Output
L.M.
1
WRITE Array Data To Direct Access Data Set
Direct Access Arrays
L.M
DBARBKCT
WAREA
WANUMARR
2
If Number Of Blocks Is Greater Than Block Count, Truncate
WAREA
WANUMARR
Return To Figure 2-157 (1 Of 6)

Figure 2-157 (5 Of 6) - WRDDSCTL - Write Array Data Control Routine

Figure 2-157 (6 of 6).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	WRITE the direct access array data to the direct access data set.		DPPXDBDA
2	If the number of blocks written is greater than the array block count, truncate all excess data blocks. The message, BLOCK COUNT EXCEEDED, is written to SYSPRINT.	DPPXDB54I	DPPXDBDA

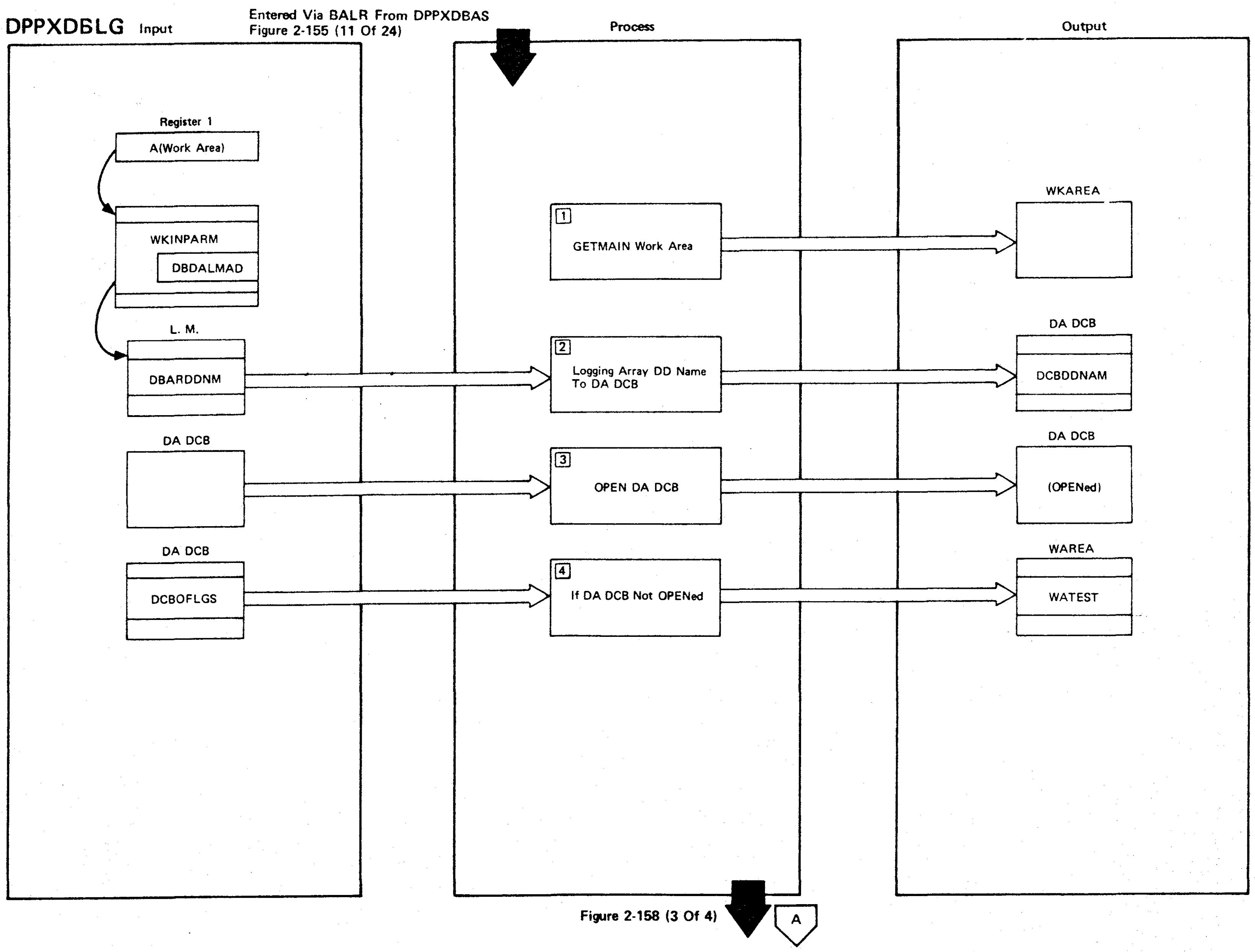


Figure 2-158 (1 Of 4) - Data Base Logging Array Formatter

Figure 2-158 (2 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	GETMAIN is used to obtain a register save area and work area.		DPPXDBLG
2	The DDNAME of the logging array data set is moved to the direct access DCB.		DPPXDBLG
3	The direct access DCB is opened.		DPPXDBLG
4	If the direct access DCB does not open, test mode is entered, and the UNABLE TO OPEN DDNAME message is written to SYSPRINT.	DPPXDB25I	DPPXDBLG

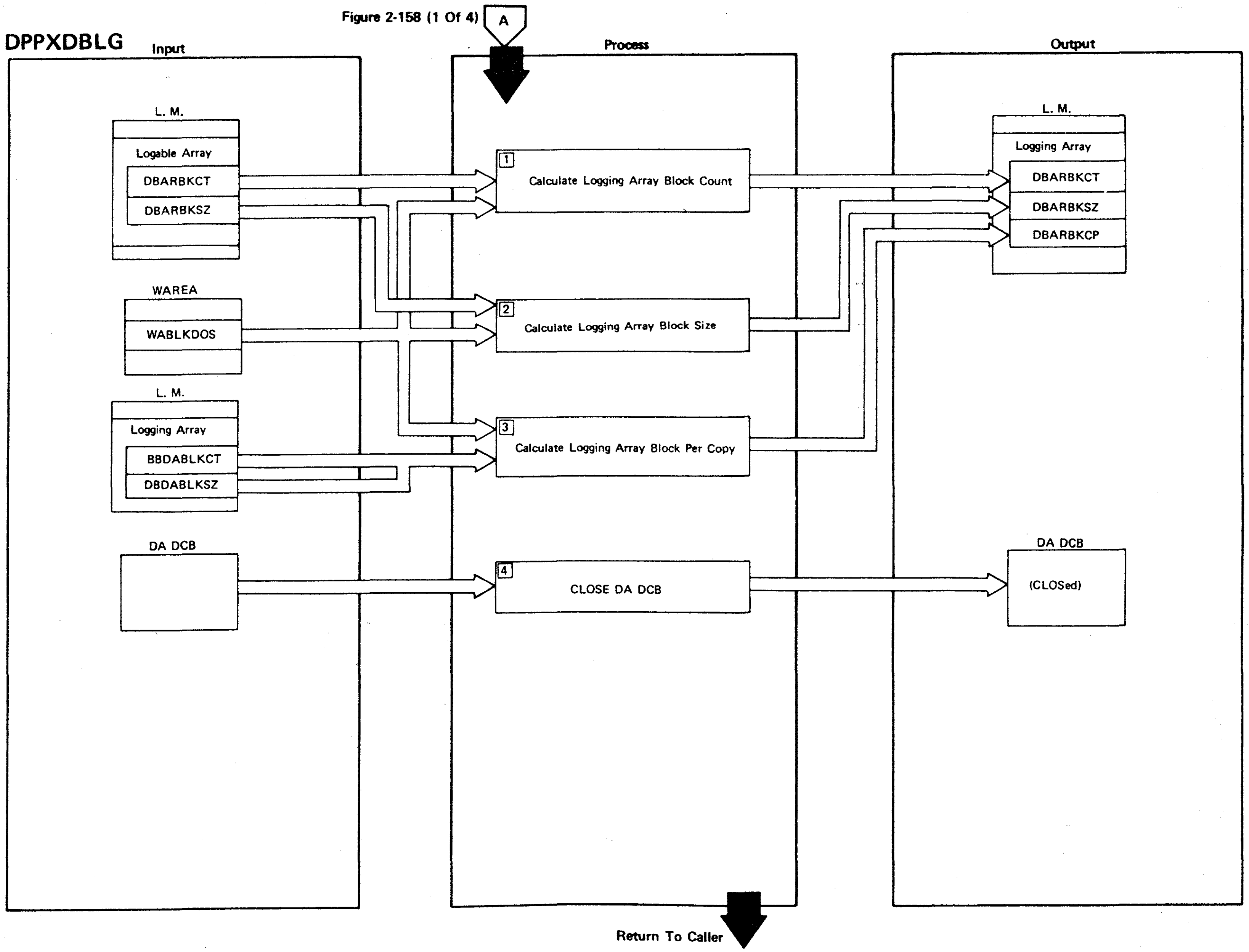


Figure 2-158 (3 Of 4) - Data Logging Array Formatter

Figure 2-158 (4 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Calculate the block count of the logging array.		DPPXDBLG
2	Calculate the blocksize of the logging array.		DPPXDBLG
3	Calculate the number of logging array blocks required to contain one copy of the loggable array.		DPPXDBLG
4	CLOSE the opened direct access DCB.		

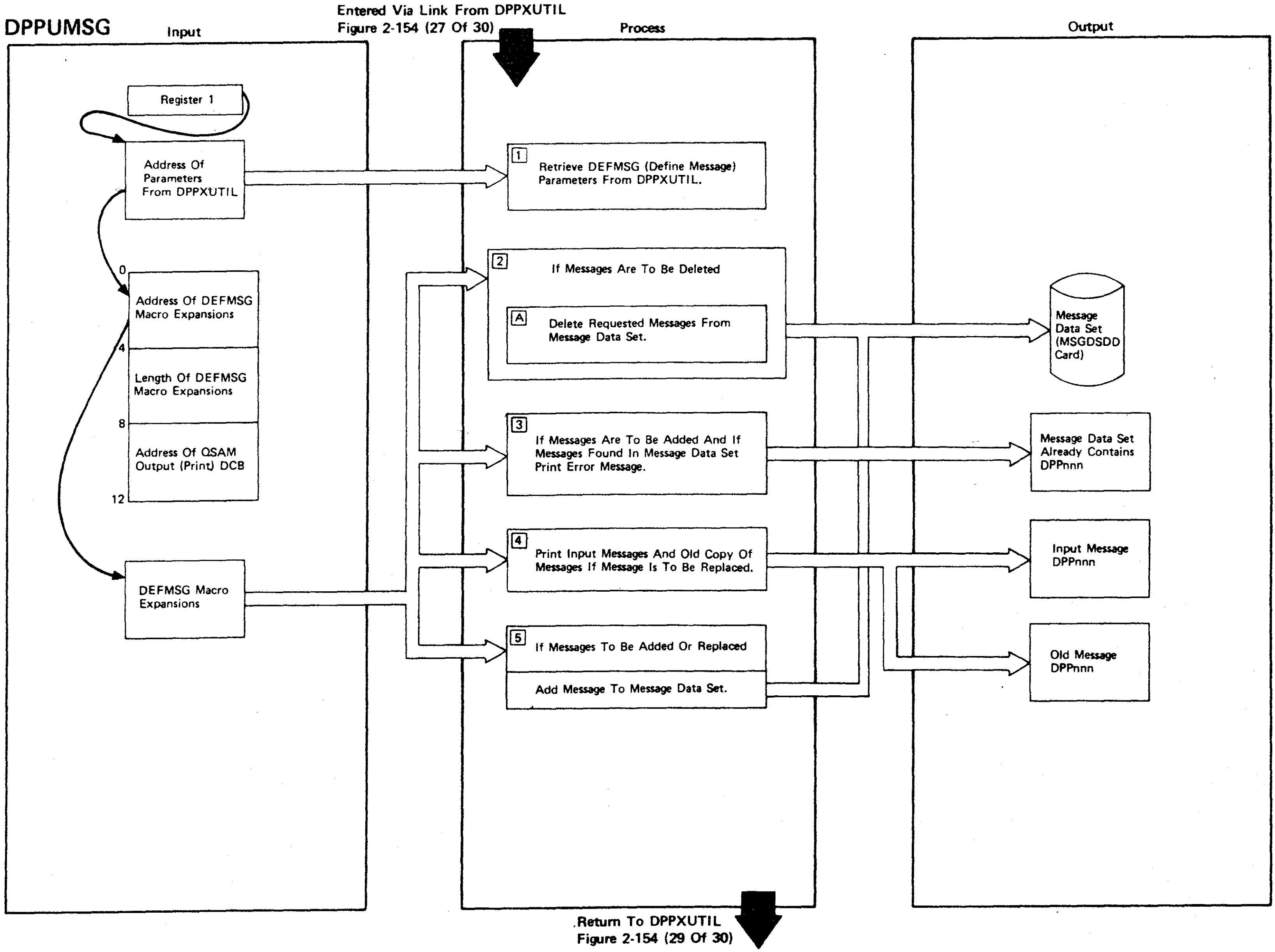
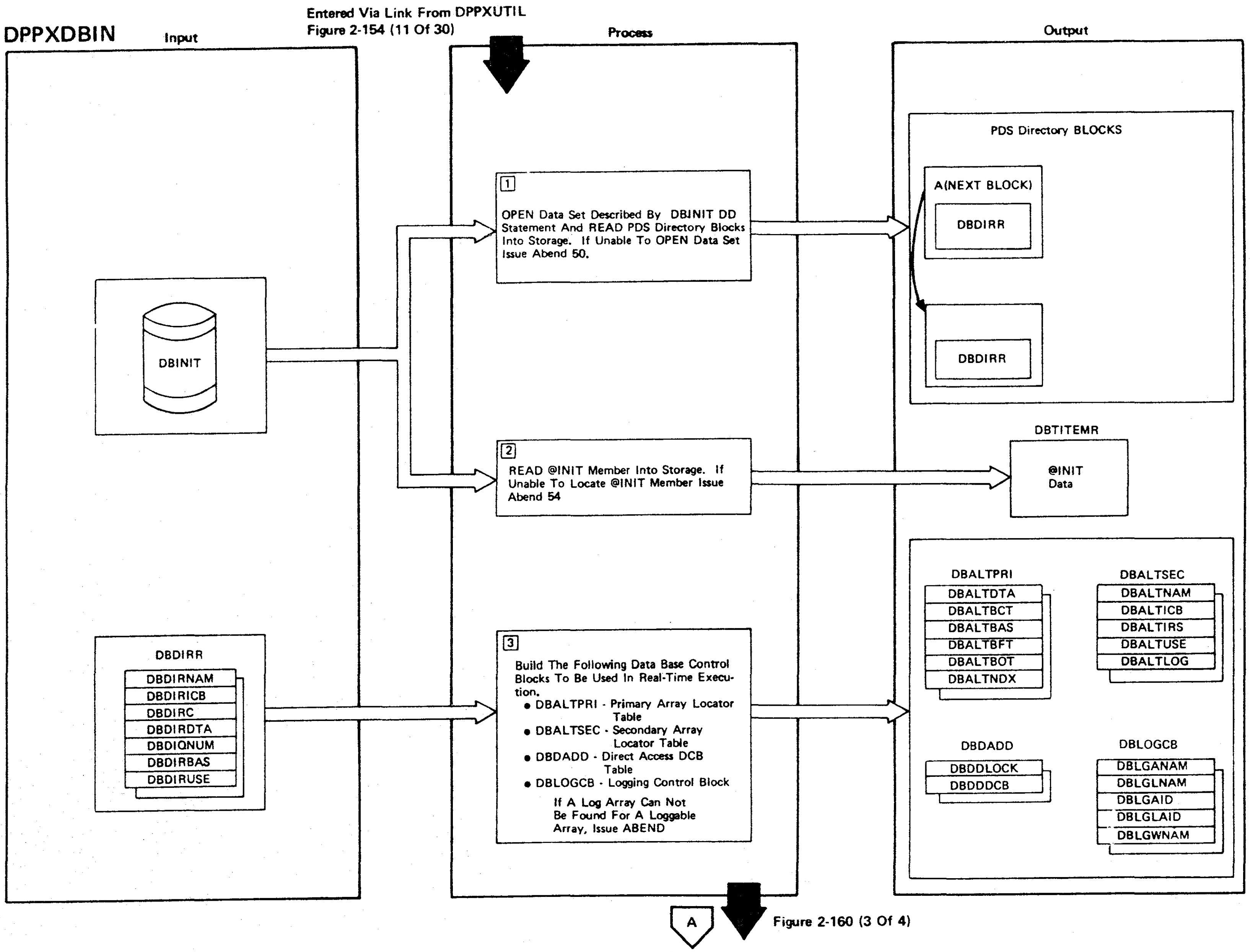

DPPUMSG
Input
Entered Via Link From DPPXUTIL
Figure 2-154 (27 Of 30)
Process
Output
Register 1
Address Of Parameters From DPPXUTIL
0
Address Of DEFMSG Macro Expansions
4
Length Of DEFMSG Macro Expansions
8
Address Of QSAM Output (Print) DCB
12
DEFMSG Macro Expansions
1
Retrieve DEFMSG (Define Message) Parameters From DPPXUTIL.
2
If Messages Are To Be Deleted
A
Delete Requested Messages From Message Data Set.
3
If Messages Are To Be Added And If Messages Found In Message Data Set Print Error Message.
4
Print Input Messages And Old Copy Of Messages If Message Is To Be Replaced.
5
If Messages To Be Added Or Replaced
Add Message To Message Data Set.
Message Data Set (MSGDSDD Card)
Message Data Set Already Contains DPPnnn
Input Message DPPnnn
Old Message DPPnnn
Return To DPPXUTIL
Figure 2-154 (29 Of 30)

Figure 2-159 (1 Of 2) - Message Final Phase Processor

Figure 2-159 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The address of the parameters passed by the offline utility (DPPXUTIL) will be retrieved from register 1.		DPPUMSG
2	If messages are to be deleted, delete the requested message from the message data set.		DPPUMSG DPPUMSG1
3	When messages are requested to be added to the existing message data set, an error message, MESSAGE DATA SET ALREADY CONTAINS DPPnnn, will be printed.		DPPUMSG2
4	All entered messages (DEFMSG expansions) will be printed. If a message is to be replaced and it already exists on the message data set, the old copy of the message will be printed.		DPPUMSG2
5	The ADD and REPLACE options cause all entered messages (DEFMSG expansions) to be added to the message data set.		DPPUMSG2



1 OPEN Data Set Described By DBINIT DD Statement And READ PDS Directory Blocks Into Storage. If Unable To OPEN Data Set Issue Abend 50.

2 READ @INIT Member Into Storage. If Unable To Locate @INIT Member Issue Abend 54

3 Build The Following Data Base Control Blocks To Be Used In Real-Time Execution.

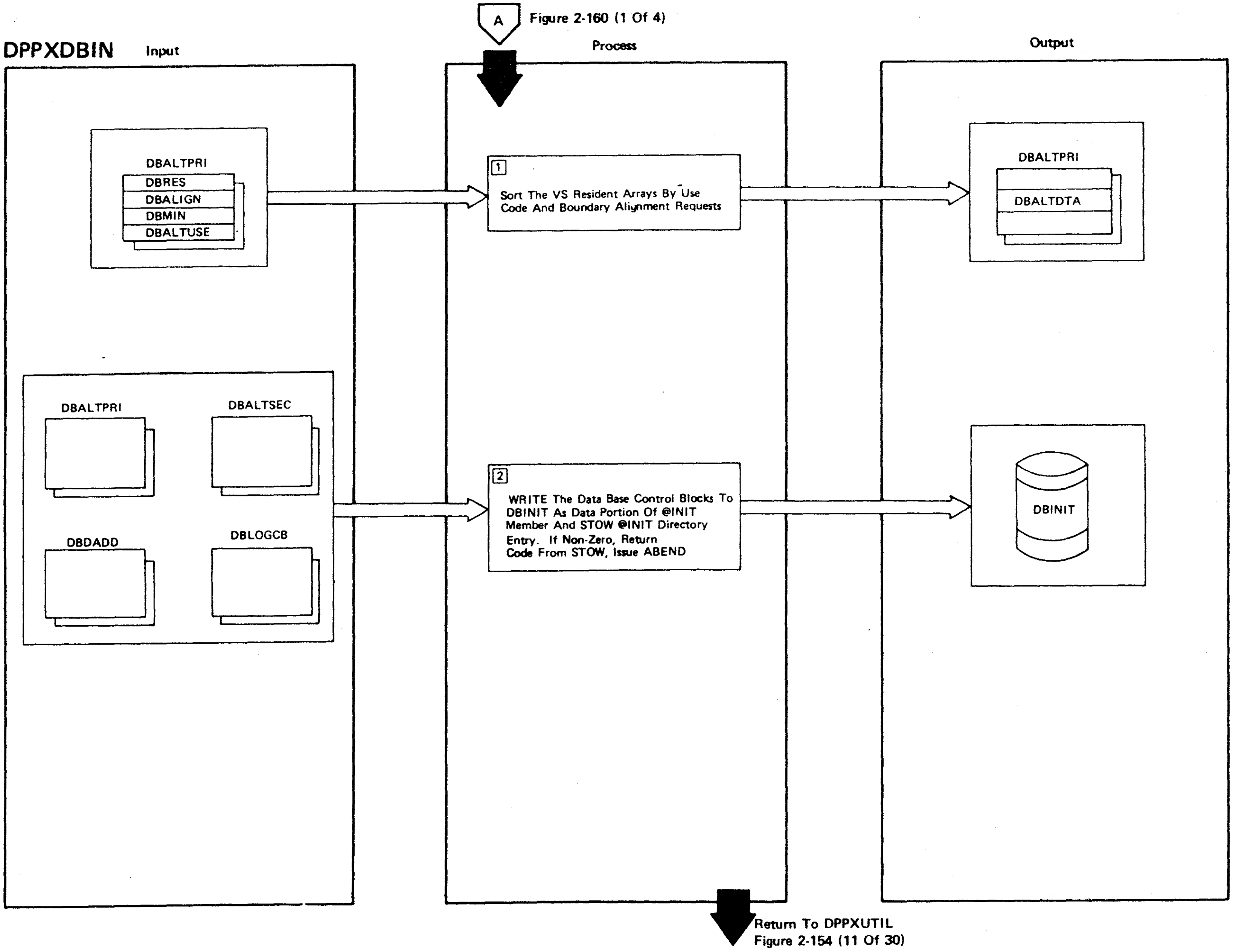
- DBALTPRI - Primary Array Locator Table
- DBALTSEC - Secondary Array Locator Table
- DBDADD - Direct Access DCB Table
- DBLOGCB - Logging Control Block

If A Log Array Can Not Be Found For A Loggable Array, Issue ABEND

Figure 2-160 (1 Of 4) - Build Data Base Initialization Array

Figure 2-160 (2 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The user-defined partitioned data set defined by the DBINIT DD statement contains data base arrays as members.	USER 50	DPXDBIN6
2	The @INIT member must have been built previously by the data base final phase processor, DPPXDBAT.	USER 52	DPXDBIN1
3	The information from the directory entries of the members (arrays) and the data associated with each directory is used to construct the Primary and Secondary Array Locator Tables for each array, the Direct Access DCB Table for each direct access array, and a Logging Control Block for each loggable array. If a log array cannot be located for each loggable array, the job step is ABENDed with code 53.	USER 53	DPXDBIN2



1 Sort The VS Resident Arrays By Use Code And Boundary Alignment Requests

2 WRITE The Data Base Control Blocks To DBINIT As Data Portion Of @INIT Member And STOW @INIT Directory Entry. If Non-Zero, Return Code From STOW, Issue ABEND

Figure 2-160 (3 Of 4) - Build Data Base Initialization Array

Figure 2-160 (4 of 4).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The VS resident arrays are arranged by a use code (1 to 7), for paging consideration. Within each use code, the arrays that require a page boundary are sorted. Then the arrays that request minimum boundary crossovers and the other arrays are filled into the slots left by the arrays with page boundary requests to prevent core fragmentation. The relative displacement of each array is placed in the Primary Array Locator table.		DPXDBIN3
2	The data base control blocks are written to the DBINIT data set. The Secondary and Primary Array Locator Tables are grouped together to be read into protected storage during real-time execution and the Directed Access DCB table and the Logging Control block are grouped together to be read into user storage. If the STOW receives a nonzero return code, the job step is ABENDed.	USER 51	DPXDBIN4

Intentionally Blank

Data Base Compress

The data base BDAM data set compress program is used to regain direct access space rendered unusable through the normal process of updating the data base through the offline utility. All usable data is moved from a data base BDAM data set into a utility data set. Then, starting at the front of the data base data set, the data is moved back into the data set, in contiguous records, compressing out unused space.

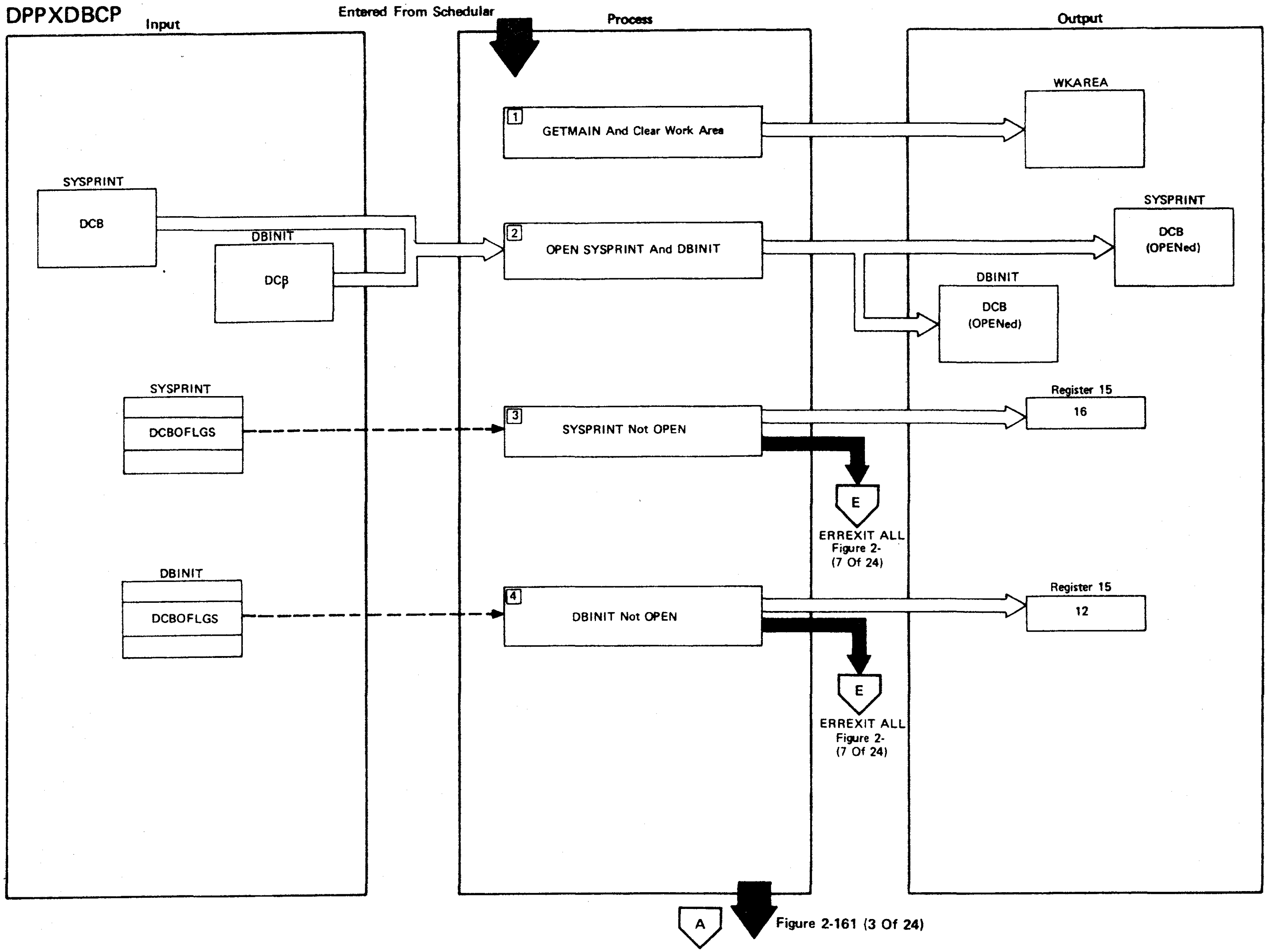


Figure 2-161 (1 Of 24) - Main Control Routine

Figure 2-161 (2 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	GETMAIN is used to obtain storage area for register save area and for work space. The area is cleared to contain all binary zeros.		DPPXDBCP
2	The SYSPRINT and DBINIT data sets are opened.		DPPXDBCP
3	If SYSPRINT does not open, register 15 is set to 16, and error exit processing is performed.		DPPXDBCP
4	If DBINIT does not open, register 15 is set to 12, the message UNABLE TO OPEN DBINIT is written to SYSPRINT, and error exit processing is performed.	DPPXDB35I	DPPXDBCP

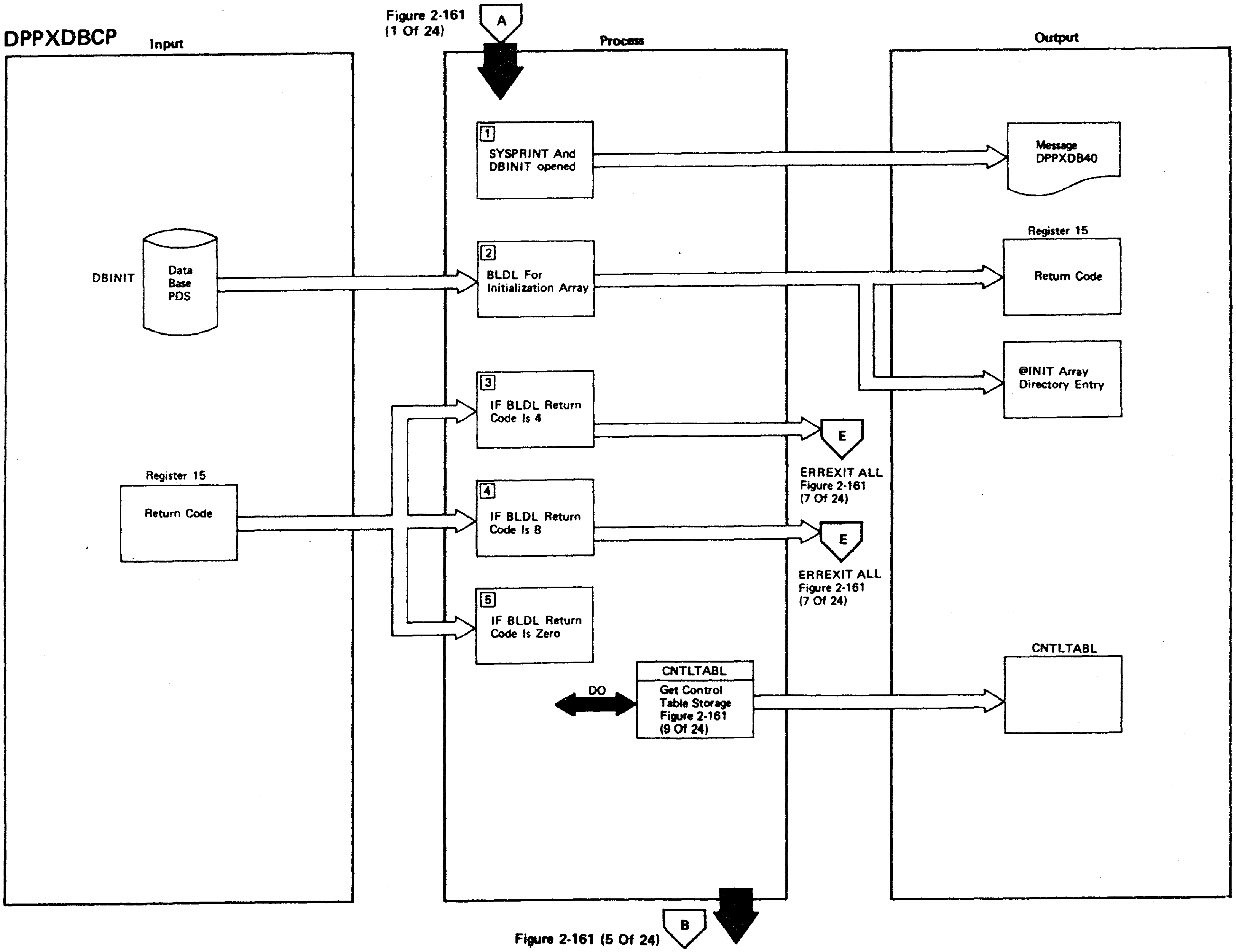


Figure 2-161 (3 Of 24) - Main Control Routine

Figure 2-161 (4 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	If SYSPRINT and DBINIT are successfully opened, the DATA BASE COMPRESS STARTED message is written to SYSPRINT.	DPPXDB40I	DPPXDBCP
2	BLDL is used to locate the data base initialization array to determine if DBINIT defines a valid data base PDS. A return code is returned in register 15.		DPPXDBCP
3	Return code 4 indicates that the initialization array could not be found. The INVALID DATA BASE DATA SET message is written to SYSPRINT, and error exit processing is performed.	DPPXDB36I	DPPXDBCP
4	Return code 8 indicates an I/O error from BLDL. The PERMANENT I/O ERROR message is written to SYSPRINT, and error exit processing is performed.	DPPXDB37I	DPPXDBCP
5	Return code 0 indicates that DBINIT is a valid data base PDS. The CNTLTABL routine calculates the size and GETMAINs the storage for the Compress Control Table.		DPPXDBCP

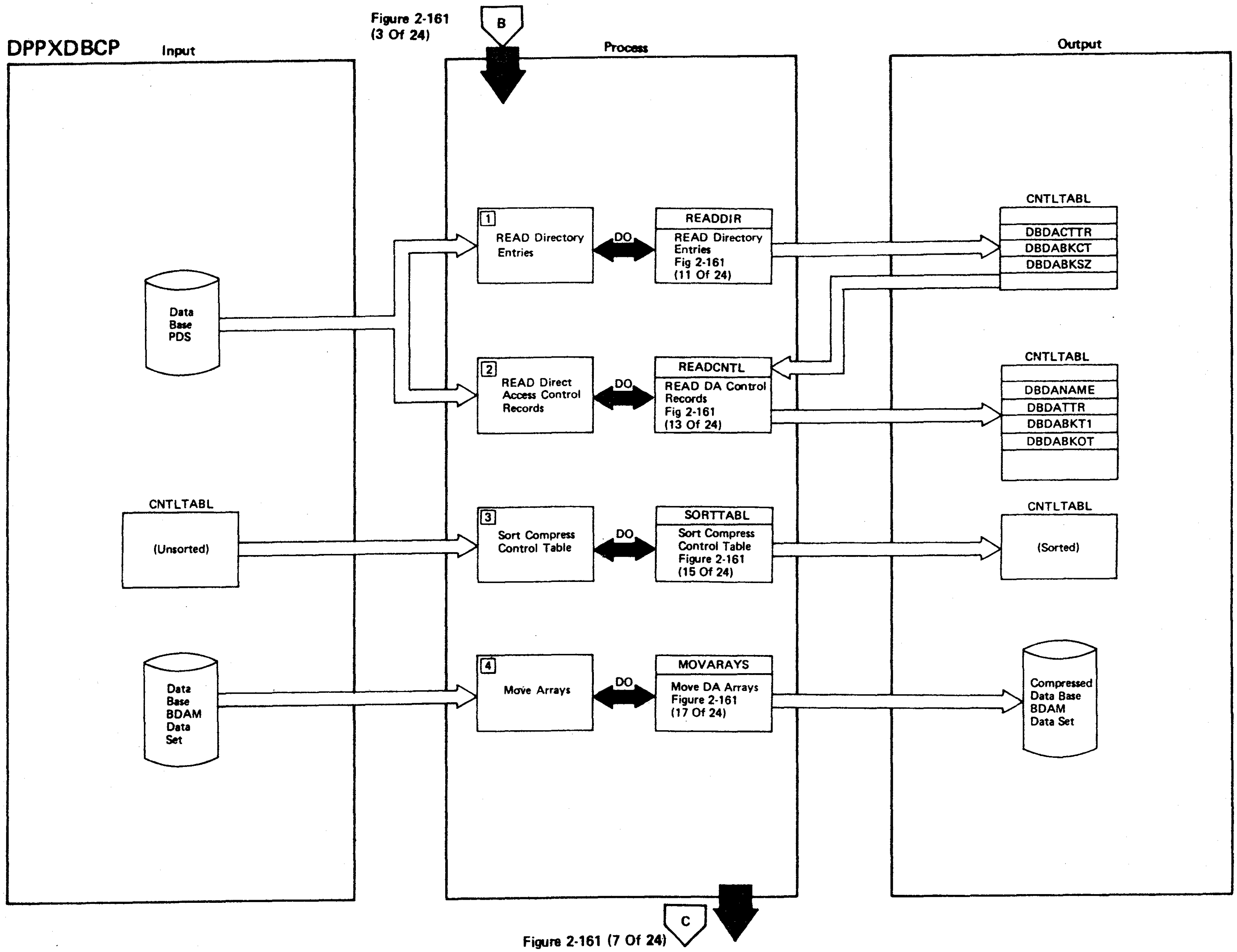


Figure 2-161 (5 Of 24) - Main Control Routine

Figure 2-161 (6 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Directory entries are read from the data base PDS. Data is moved to the Compress Control Table from the directory entry for each direct access resident array.		DPPXDBCP
2	The direct access control record for each direct access resident array is moved into the Compress Control Table entry that contains directory data for that array.		DPPXDBCP
3	The Compress Control Table entries are sorted by the DDNAME contained in the direct access control record portion of the Compress Control Table.		DPPXDBCP
4	Data base BDAM data sets are compressed by moving direct access resident arrays.		DPPXDBCP

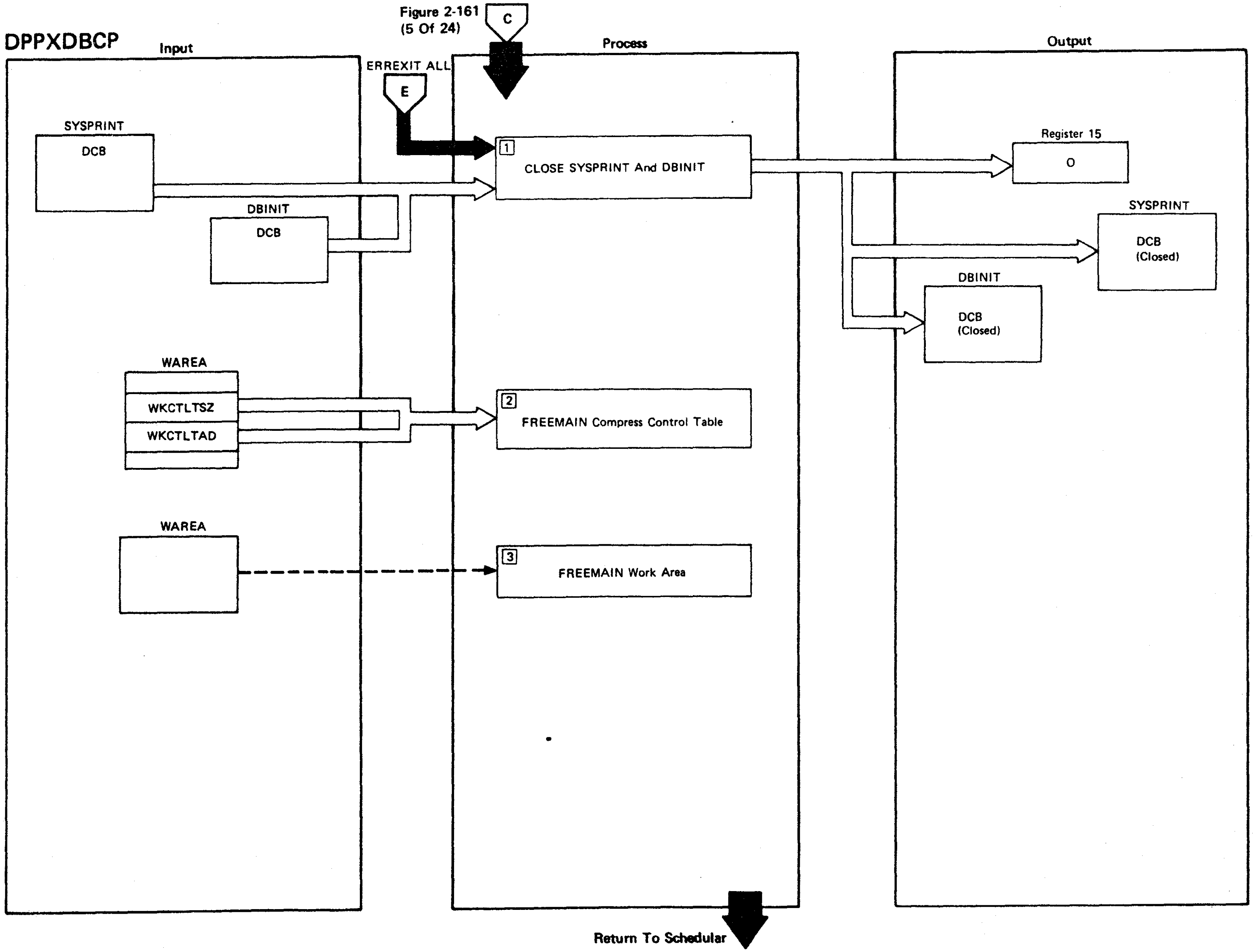


Figure 2-161 (7 Of 24) - Main Control Routine

Figure 2-161 (8 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The DATA BASE COMPRESS COMPLETED message is written to SYSPRINT, return code 0 is set in register 15, and SYSPRINT and DBINIT DCBs are closed.	DPPXDB4I	DPPXDBCP
2	The Compress Control Table area of storage is freed.		DPPXDBCP
3	The work area and the register save area are freed.		DPPXDBCP

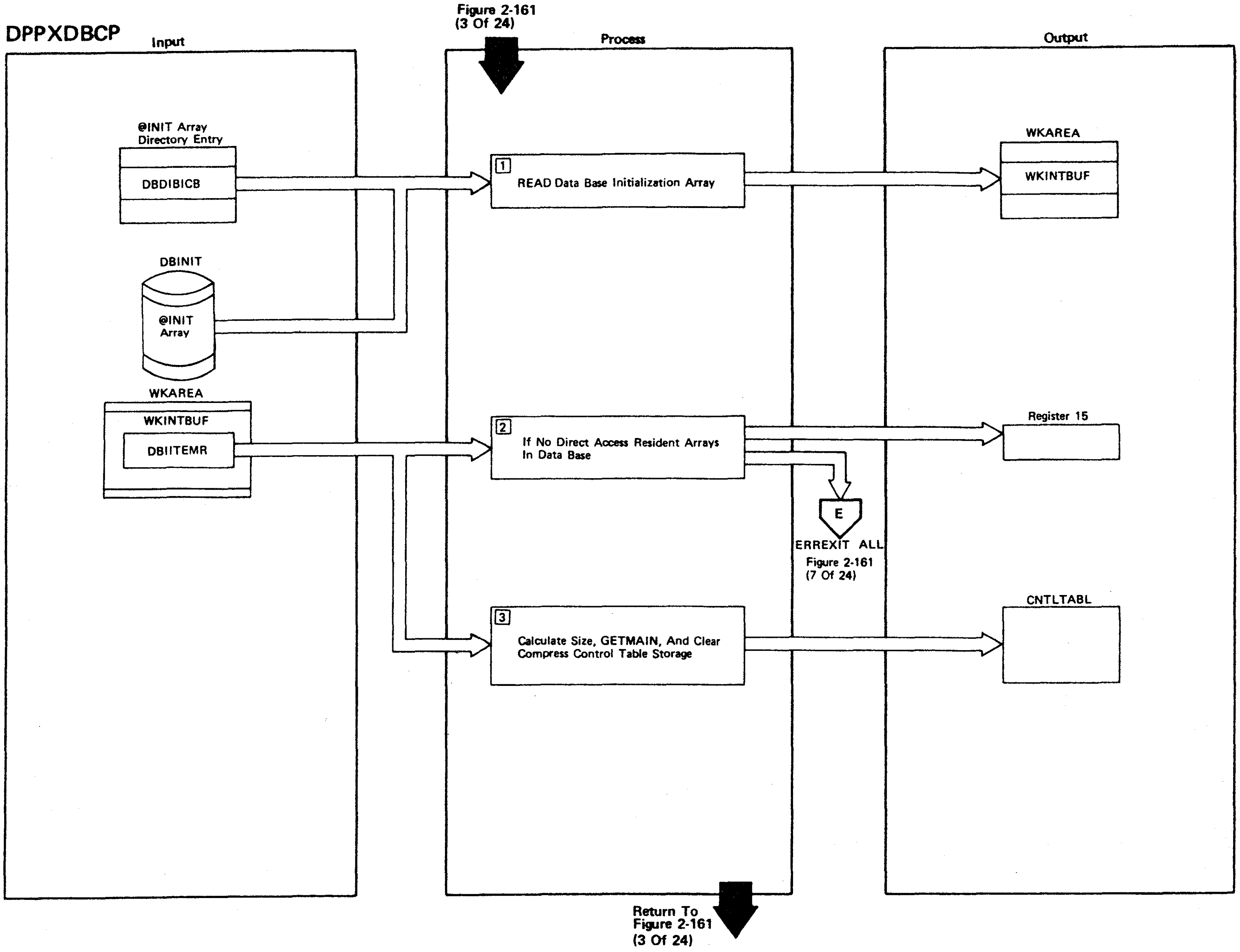


Figure 2-161 (9 Of 24) - CNTLTABL - Get Compress Control Table

Figure 2-161 (12 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	READ directory blocks from data base PDS into a directory block buffer.		DPPDXDBCP
2	The directory entries in each directory block are searched for direct access resident arrays.		DPPXDBCP
3	The direct access control record TTR, the array block count, and the array blocksize are moved from the directory entry to the Compress Control Table.		DPPXDBCB

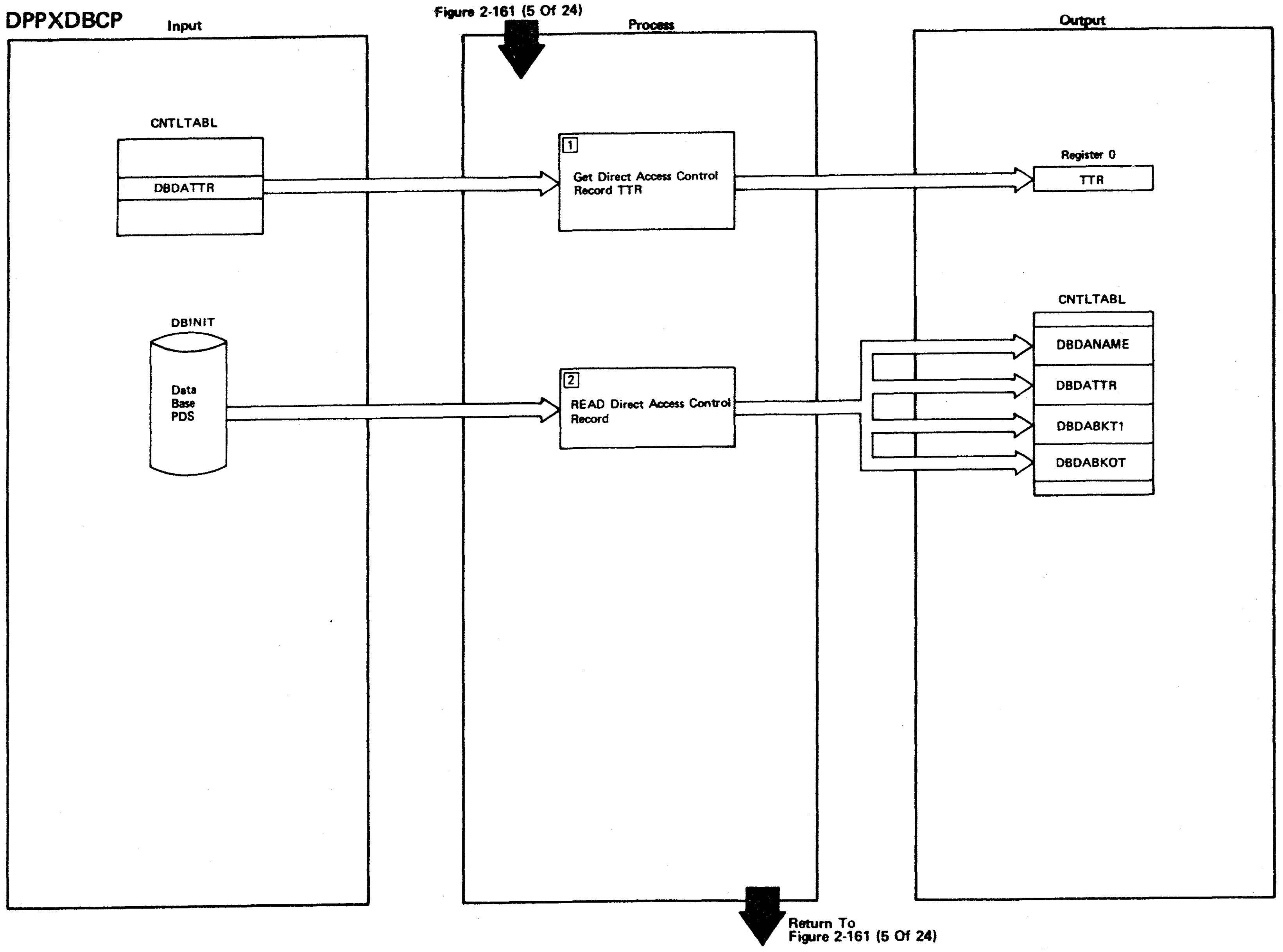


Figure 2-161 (13 Of 24) - READCNTL - Read Direct Access Control Records

Figure 2-161 (14 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Get the direct access control record TTR from the Compress Control Table.		DPPXDBCP
2	READ the direct access control record from the data base PDS and put it into the Compress Control Table.		DPPXDBCP

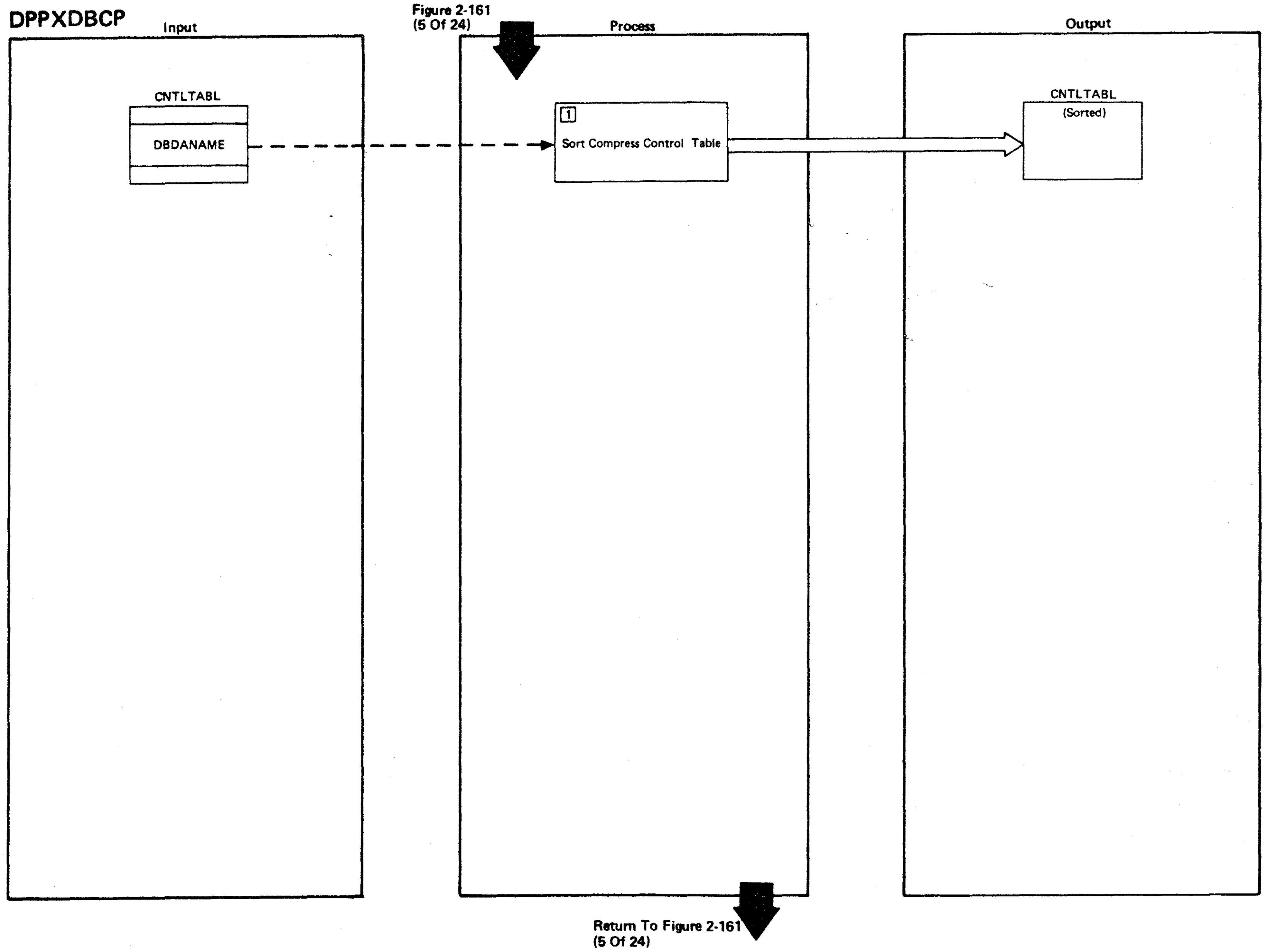


Figure 2-161 (15 Of 24) - SORTTABL - Sort Compress Control Table

Figure 2-161 (16 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The Compress Control Table entries are sorted in ascending alphanumeric sequence on the DDNAME field (DBDANAME).		DPPXDBCP

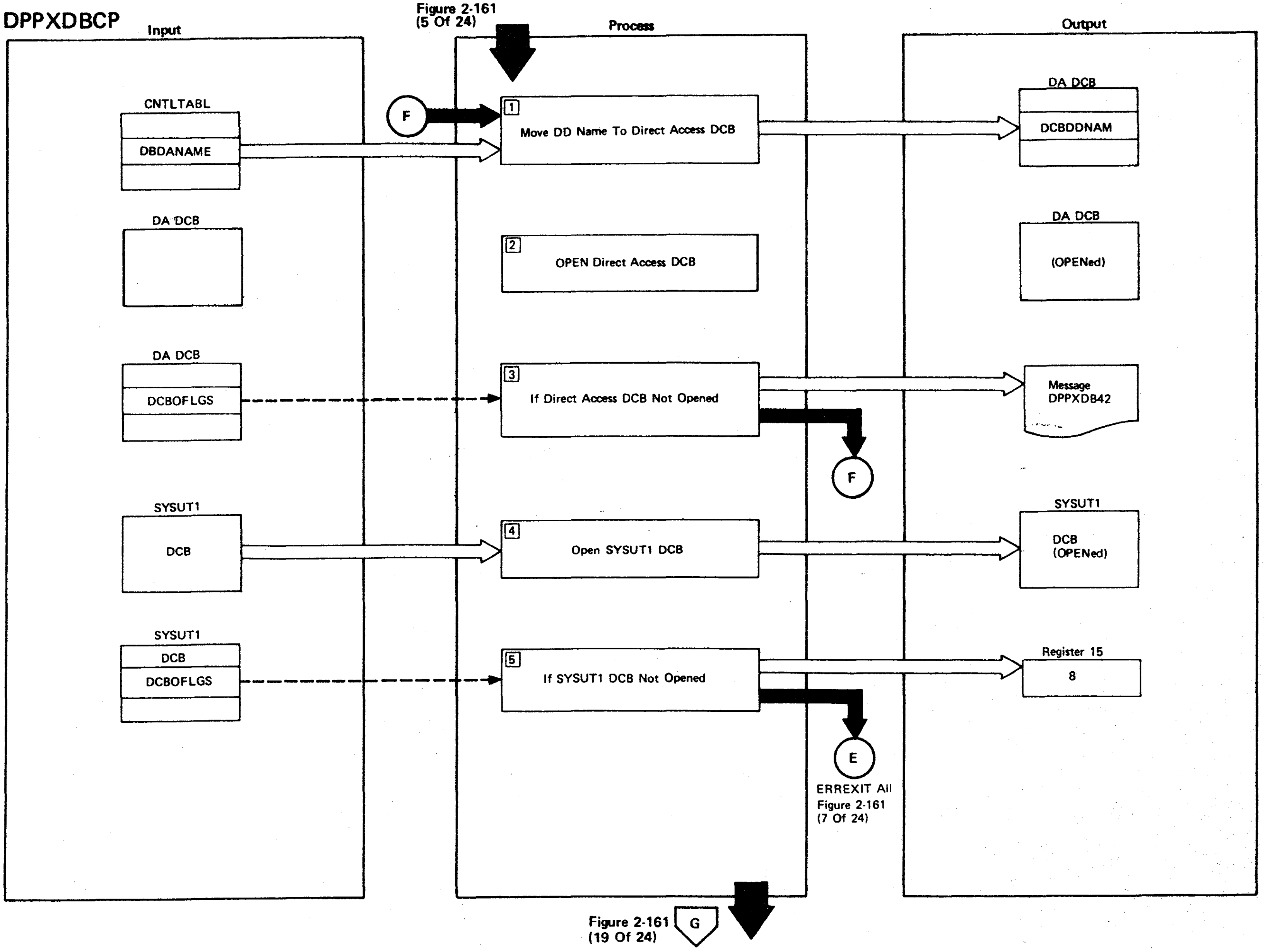


Figure 2-161 (17 Of 24) - MOVARAYS - Move Direct Access Arrays

Figure 2-161 (18 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The DDNAME is moved from the Compress Control Table to the direct access DCB.		DDPXDBCP
2	The direct access DCB is opened.		DPPXDBCP
3	If the OPEN of the direct access DCB is not successful, the UNABLE TO OPEN DD STATEMENT message is written to SYSPRINT. Processing will continue with the next Compress Control Table entry with a different DDNAME.	DPPXDB42I	DPPXDBCP
4	The SYSUT1 DCB is opened.		DPPXDBCP
5	If the OPEN of the SYSUT1 DCB is not successful, the UNABLE TO OPEN SYSUT1 message is written to SYSPRINT, return code 8 is set in register 15, and error exit processing is performed.	DPPXDB35I	DPPXDBCP

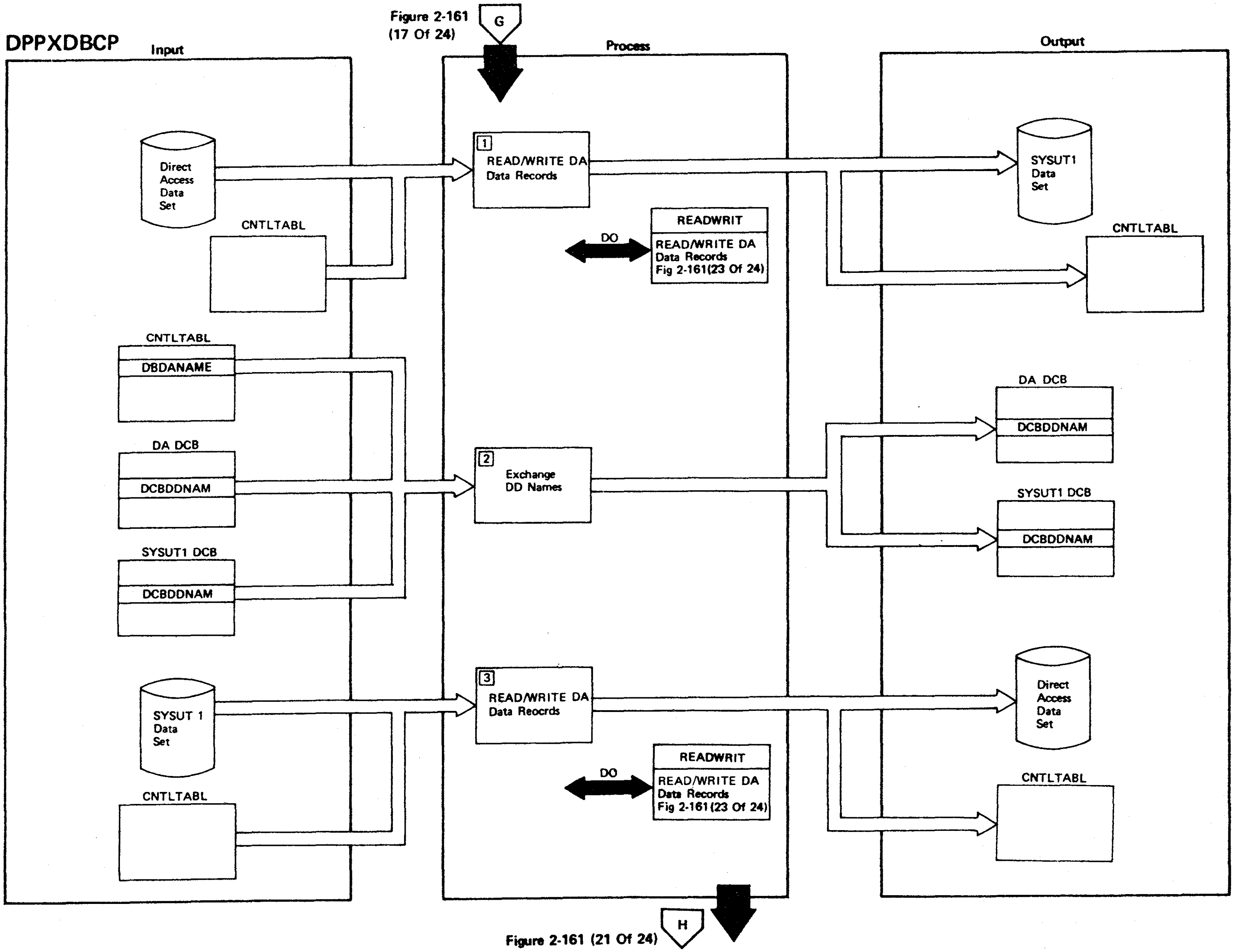


Figure 2-161 (19 Of 24) - MOVARAYS - Move Direct Access Arrays

Figure 2-161 (20 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	READ direct access array data from the DA data set and WRITE the data to the SYSUT1 data set.		DPPXDBCP
2	The DDNAMEs in the DA DCB and the SYSUT1 DCB are exchanged.		DPPXDBCP
3	READ direct access array data from the SYSUT1 data set and WRITE the data to the DA data set.		DPPXDBCP

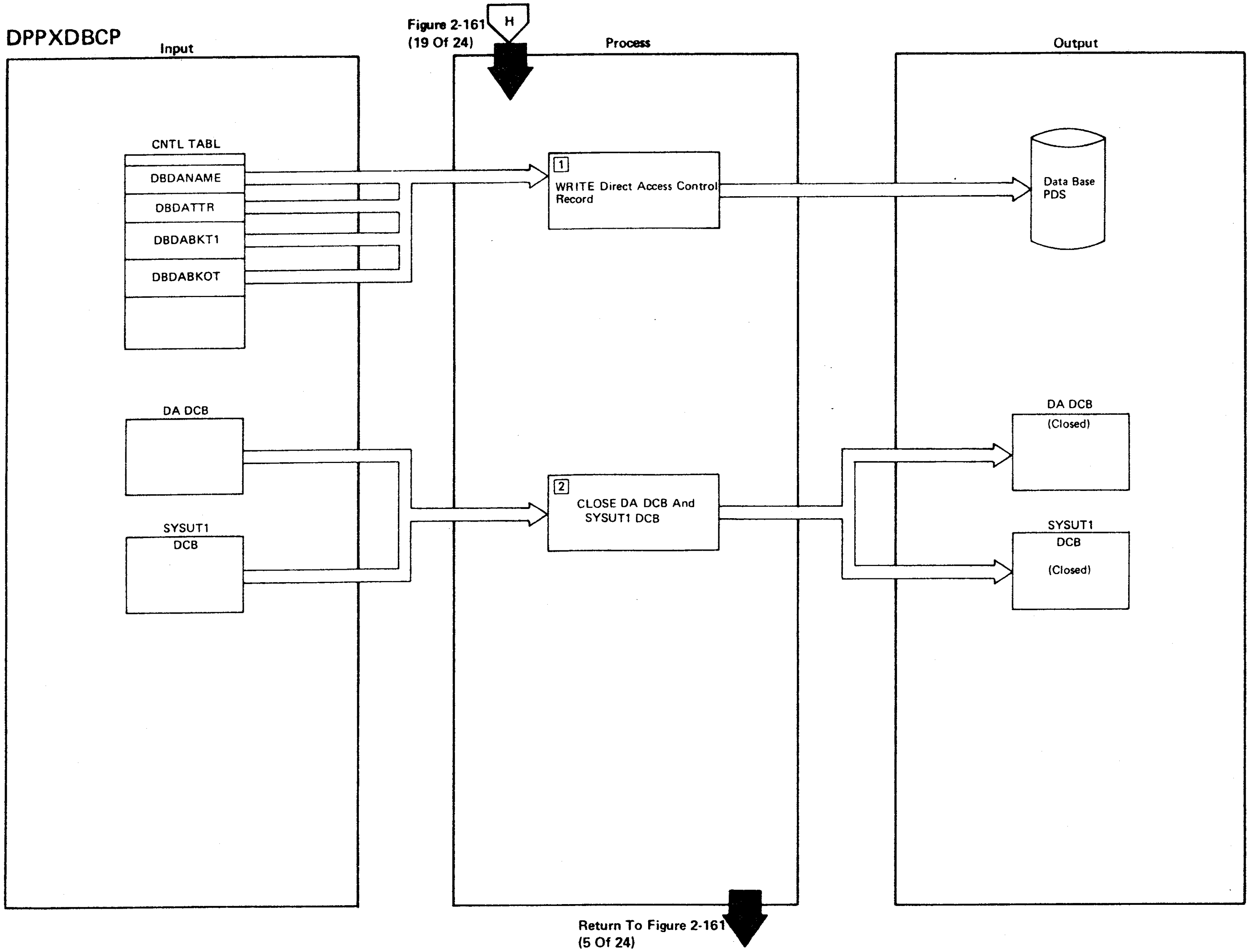


Figure 2-161 (21 Of 24) - MOVARAYS - Move Direct Access Arrays

Figure 2-161 (22 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	WRITE the updated direct access control data from the Compress Control Table to the data base PDS direct access control record.		DPPXDBCP
2	CLOSE the direct access data set DCB and the SYSUT1 DCB. The DATA SET COMPRESSED message is written to SYSPRINT.	DPPXDB39I	DPPXDBCP

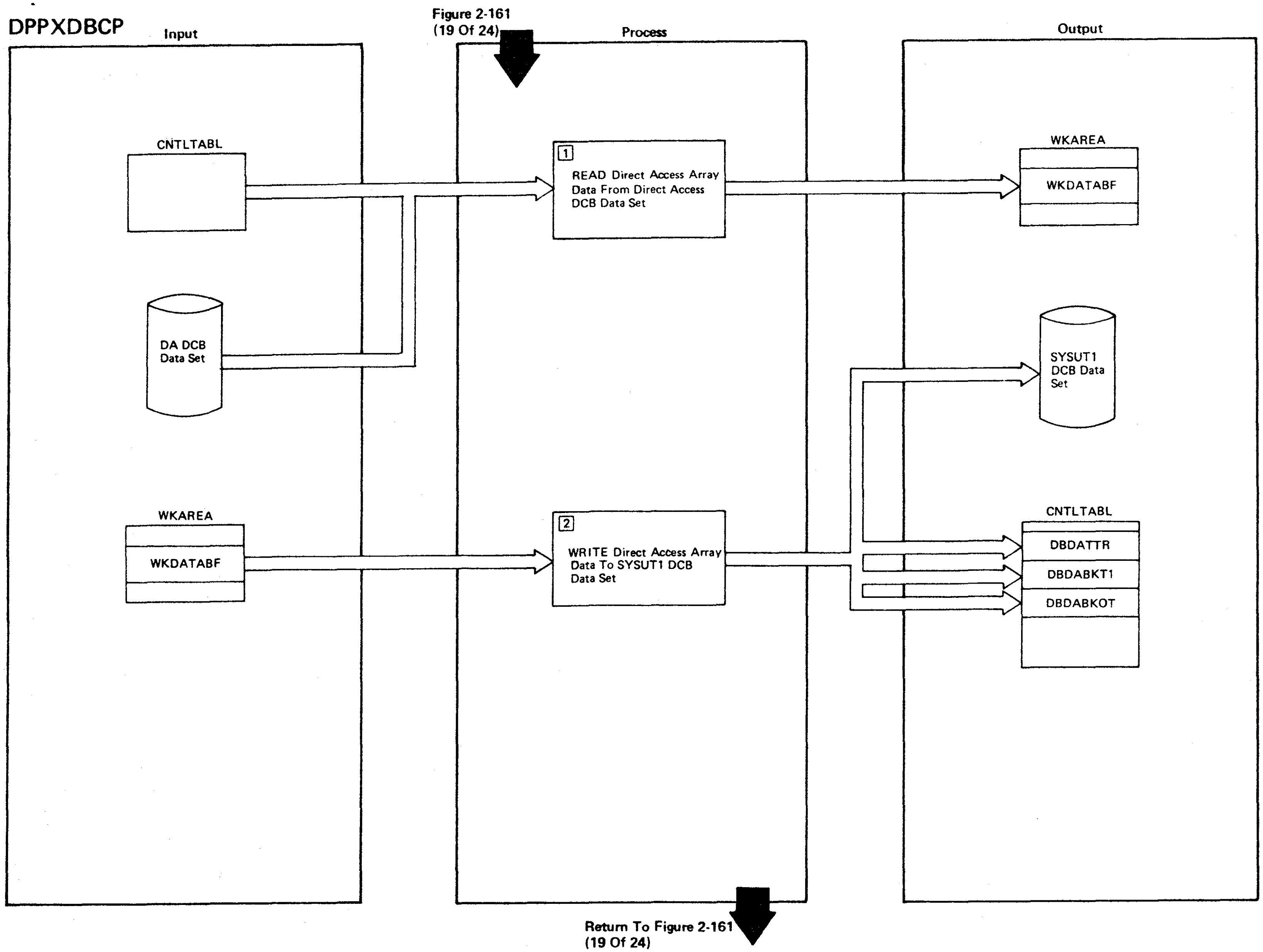


Figure 2-161 (23 Of 24) - READWRIT - Read/Write DA Data Records

Figure 2-161 (24 of 24).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Control Data is taken from the Compress Control Table and used in reading direct access array data from the data set defined by the DA DCB.		DPPXDBCP
2	The direct access array data is written to the data set defined by the SYSUT1 DCB. Direct access control data is updated in the Compress Control Table.		DPPXDBCP

Intentionally Blank

Playback Routine

The playback routine is used in an offline environment to convert, format, and print data recorded during online execution. The data played back was created using the RECORD facility of the Special Real Time Operating System.

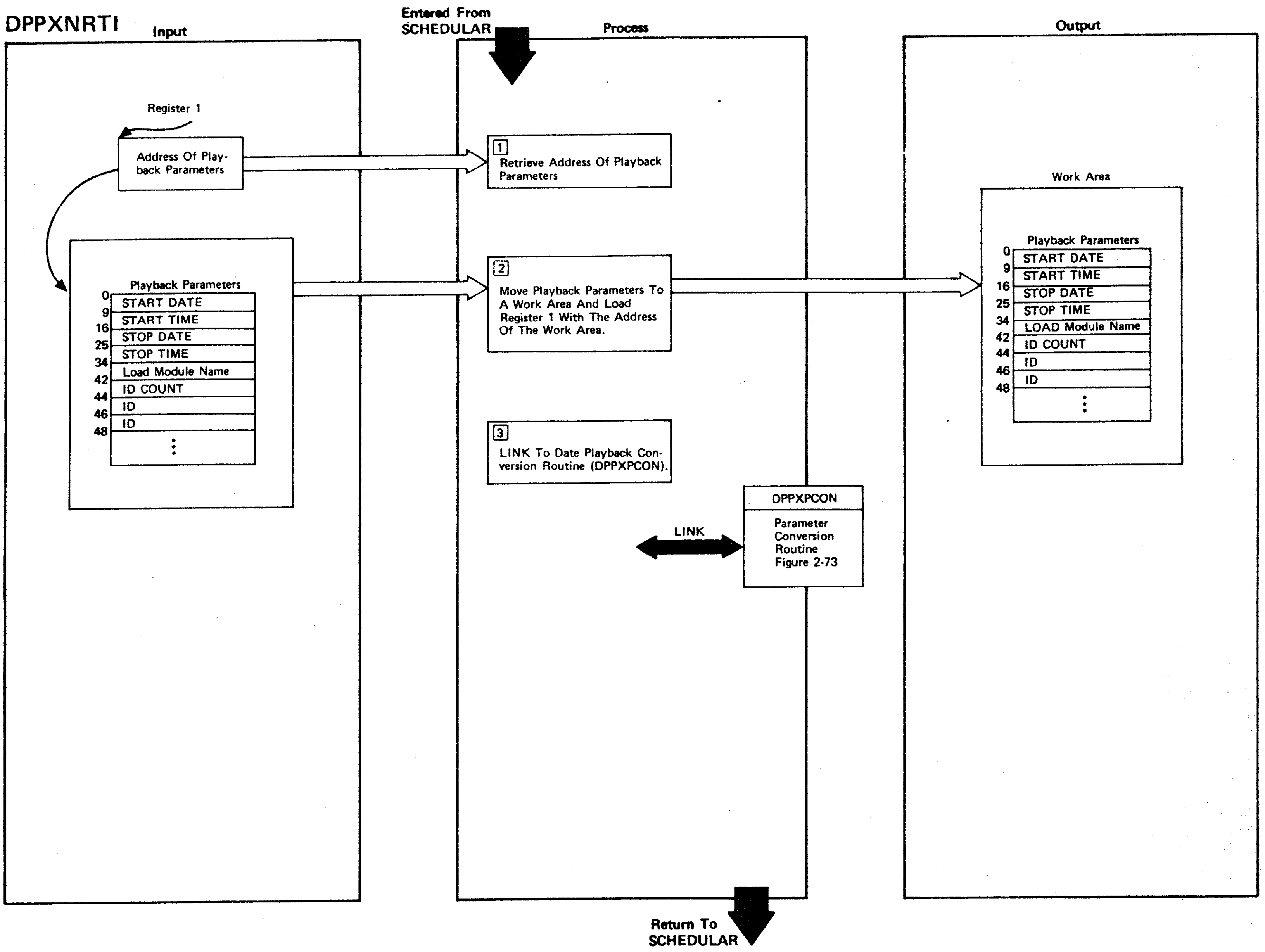


Figure 2-162 (1 Of 2) - Data Playback Non-Real Time Initialization Routine

Figure 2-162 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Retrieve the address of playback parameters from register 1.		DPPXNRTI
2	All playback parameters will be moved into a work area built by DPPXNRTI. The address of the work area will be stored in register 1.		DPPXNRTI
3	An OS/VS1 LINK macro will be issued to the data playback conversion routine (DPPXPCON).		DPPXNRTI

Intentionally Blank

SYSGEN Utility

The stage I SYSGEN Utility uses software option and hardware configurations specifications as input. The output is a printed listing and the job stream for stage II of SYSGEN. The job stream, when executed creates a Special Real Time Operating System.

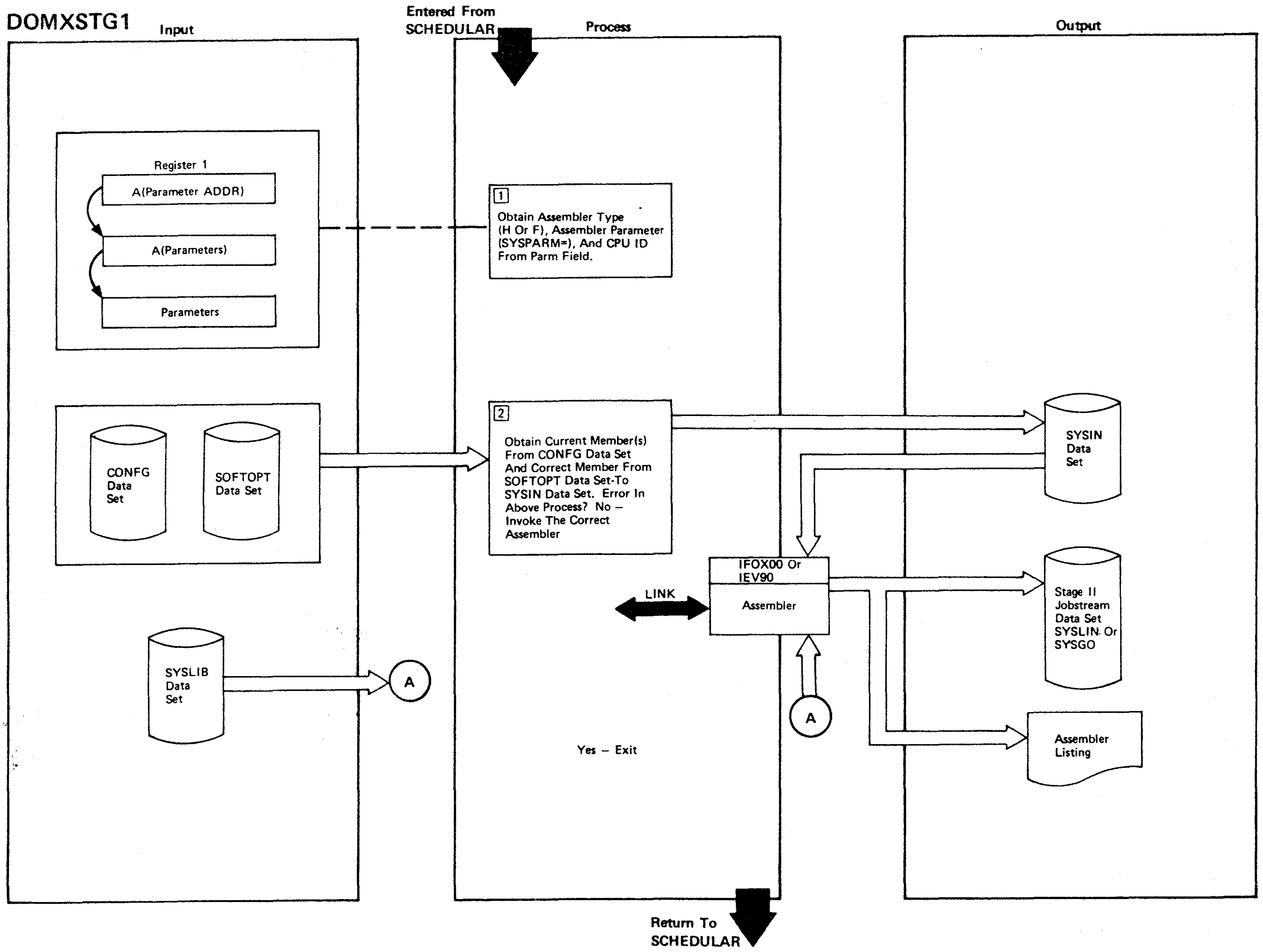

DOMXSTG1
Input
Entered From SCHEDULAR
Process
Output
Register 1
A(Parameter ADDR)
A(Parameters)
Parameters
1
Obtain Assembler Type (H Or F), Assembler Parameter (SYSPARM=), And CPU ID From Parm Field.
CONFG Data Set
SOFTOPT Data Set
2
Obtain Current Member(s) From CONFG Data Set And Correct Member From SOFTOPT Data Set-To SYSIN Data Set. Error In Above Process? No — Invoke The Correct Assembler
SYSIN Data Set
IFOX00 Or IEV90
Assembler
LINK
Stage II Jobstream Data Set SYSLIN Or SYSGO
SYSLIB Data Set
A
A
Assembler Listing
Yes – Exit
Return To SCHEDULAR

Figure 2-163 (1 Of 2) - Stage 1 SYSGEN Utility

Figure 2-163 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The CPU ID is of the form System/370xx or System/7xx where xx is any decimal number from 01 to 99. If an error is detected, issue message INVALID PARM FIELD.		DOMXSTG1
2	The correct members are: A. The member of the configuration data set named in the PARM field. B. All other configuration members named in the first or subsequent configuration member(s) CONFIGH child parameters*. C. The member of the SOFTOPT data set named in the PARM field. *Multiple CONFGH members will be used only if the System/370 Energy Management System Program Product (5740-U11) is being generated.		DOMXSTG1

Intentionally Blank

SAMPLE PROGRAMS

The Special Real Time Operating System sample programs provide a minimal test of the functioning of the Special Real Time Operating System. The primary program, DPPZSAMP, exercises the following functional areas:

Task Management

Time Management

Data Base Management

Message Handler

The secondary program, DPPSAMP1, is used to substantiate the functioning of test management and time management routines by issuing message 66 whenever entered as the result of PATCH or PTIME macro call.

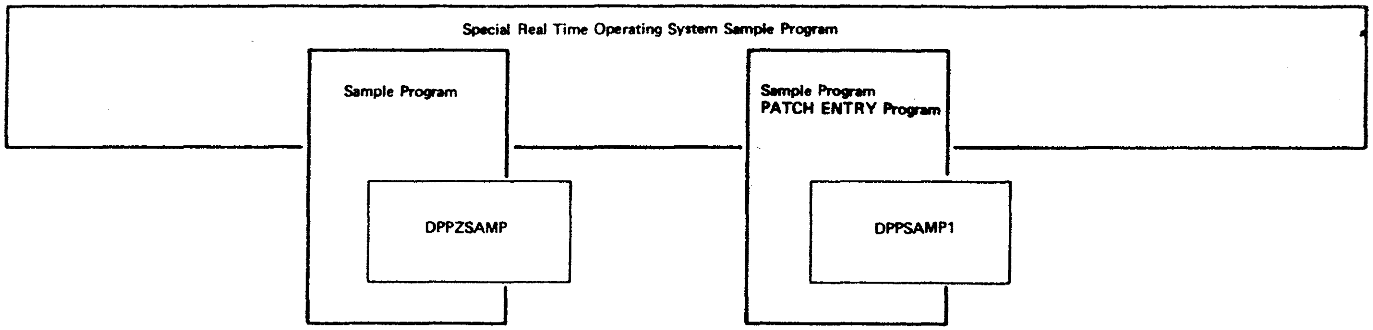


Figure 2-164 (1 of 2) Special Real Time Operating System Sample Program Overview

Intentionally Blank

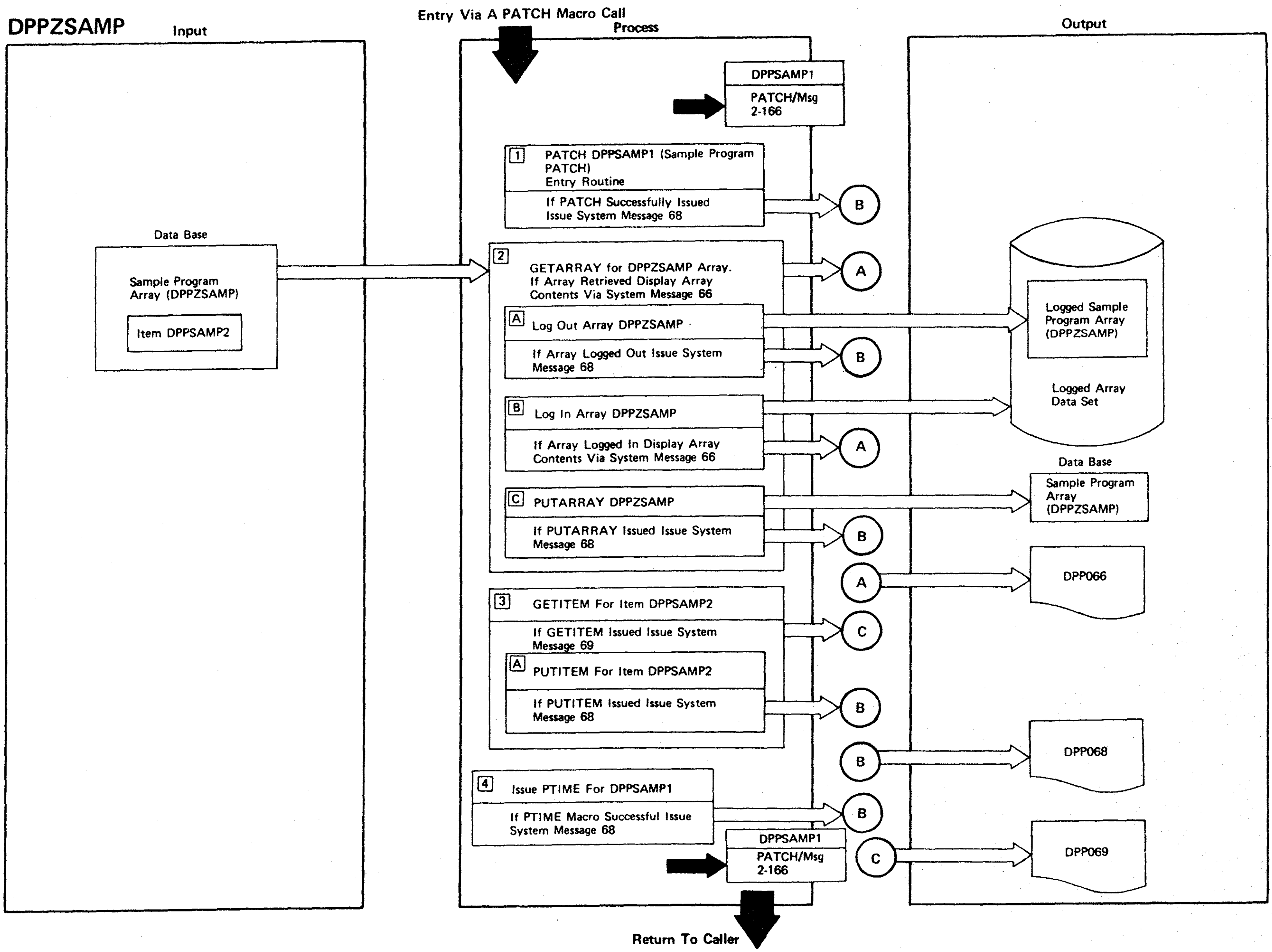

DPPZSAMP
Input
Entry Via A PATCH Macro Call
Process
Output
DPPSAMP1
PATCH/Msg
2-166
1 PATCH DPPSAMP1 (Sample Program PATCH) Entry Routine
If PATCH Successfully Issued Issue System Message 68
B
Data Base
Sample Program Array (DPPZSAMP)
Item DPPSAMP2
2 GETARRAY for DPPZSAMP Array. If Array Retrieved Display Array Contents Via System Message 66
A
A Log Out Array DPPZSAMP
If Array Logged Out Issue System Message 68
B
Logged Sample Program Array (DPPZSAMP)
Logged Array Data Set
B Log In Array DPPZSAMP
If Array Logged In Display Array Contents Via System Message 66
A
Data Base
Sample Program Array (DPPZSAMP)
C PUTARRAY DPPZSAMP
If PUTARRAY Issued Issue System Message 68
B
A
DPP066
3 GETITEM For Item DPPSAMP2
If GETITEM Issued Issue System Message 69
C
A PUTITEM For Item DPPSAMP2
If PUTITEM Issued Issue System Message 68
B
B
DPP068
4 Issue PTIME For DPPSAMP1
If PTIME Macro Successful Issue System Message 68
B
DPPSAMP1
PATCH/Msg
2-166
C
DPP069
Return To Caller

Figure 2-165 (1 Of 2) - Special Real Time Operating System Sample Program

Figure 2-165 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	The sample program will PATCH task DPPSAMP1 at entry point DPPSAMP1	DPP068I	DPPZSAMP
2	Array DPPZSAMP will be retrieved from the data base by a GETARRAY macro.	DPP066I	DPPZSAMP
	A. Array DPPZSAMP will be logged out by the PUTLOG macro.	DPP068I	
	B. Array DPPZSAMP, contents of the array logged, will be logged in by a GETLOG macro.	DPP066I	DPPZSAMP
	C. Array DPPZSAMP, contents of the array logged, will be placed back in the data base by a PUTARRAY macro.	DPP068I	
3	Item DPPSAMP2 will be retrieved from the data base by a GETITEM macro.	DPP069I	
	A. Item DPPSAMP2 contents retrieved by the GETITEM in step 3 will be placed in the data base by a PUTITEM macro.	DPP068I	DPPZSAMP
4	A PTIME macro will be issued to cause task DPPSAMP1, at entry point DPPSAMP1, to be patched three times at 1-second intervals.	DPP068I	DPPZSAMP

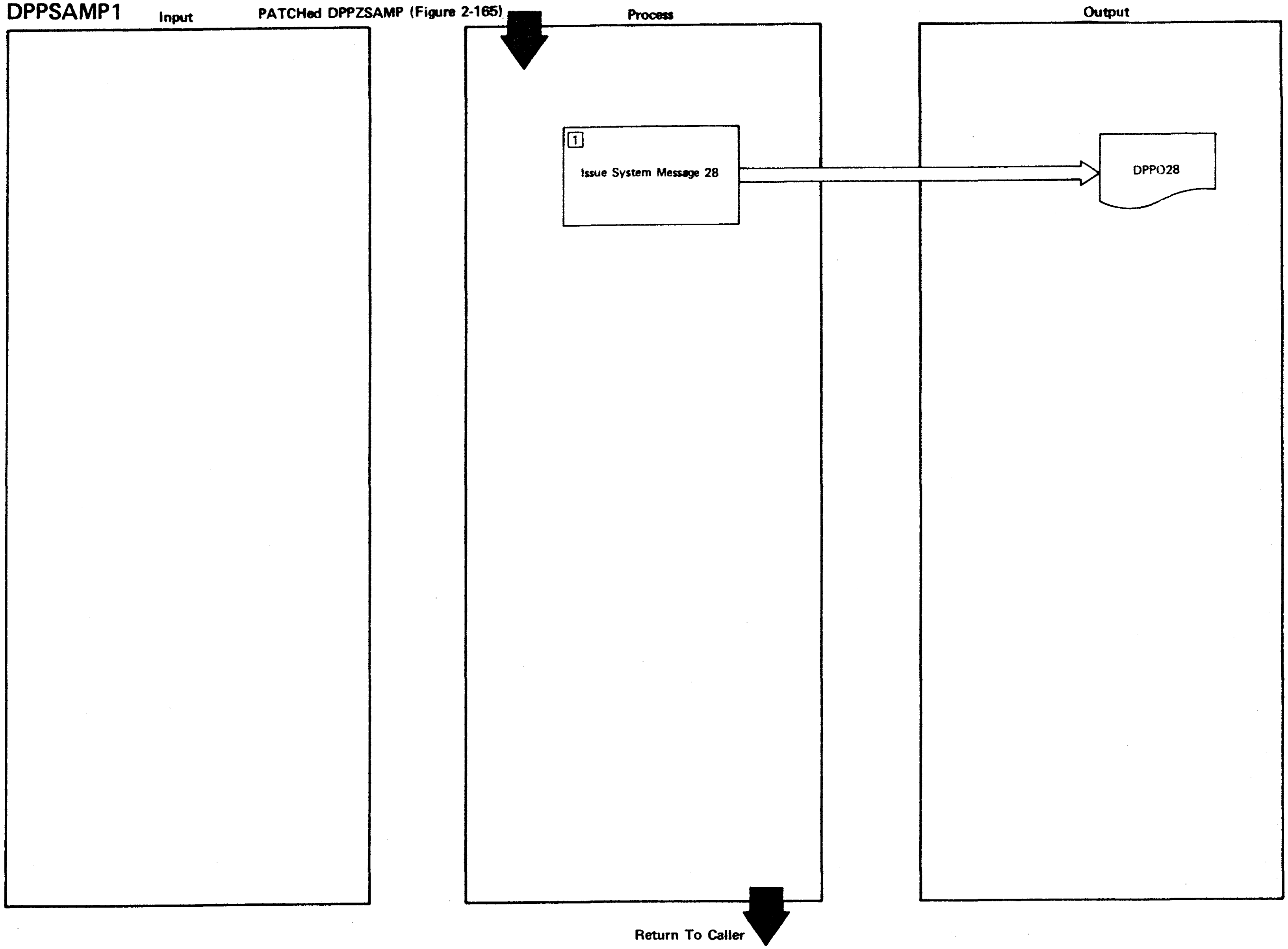


Figure 2-166 (1 Of 2) - Sample Program PATCH Entry Routine

Figure 2-166 (2 of 2).

Step	Extended Description	Messages and ABEND Codes	PDL Segment
1	Issue message 28 to indicate the sample program has been entered.	DPP028I	DPPSAMP1

Section 3. PROGRAM ORGANIZATION DESCRIPTION

The Program Organization section of the System Logic Manual describes the method of implementation of the functions described in Section 2. This is done through Program Design Language (PDL) representation of ;each member of source code that comprise the Special Real Time Operating System. PDL provides a more detailed insight into the logic flow of the individual programs and closely resembles the actual program structure. This section is intended to be an intermediary step and to aid in the transition from the functional overview of the logic descriptions (Section 2) to the program listings themselves. The HIPO charts in Section 2 can be used to tie a function in question to one or more program segments. The PDL charts in this section can then be used to pinpoint the area of concern within the referenced segment(s). Neither the HIPO charts nor the PDL charts are intended to be a subset of the other; nor is either a one-to-one mapping of the other. They support each other and should be used together to fully understand the logic flow of the program.

In some cases, several members of source code are combined, by use of COPY statements, into one assembly. A cross-reference of source members that are COPYed into base CSECTs is provided in Appendix A. A cross-reference of module names to HIPO and PDL charts is provided in Appendix C. A cross-reference of ABEND codes and message numbers to module names can be found in the Description and Operations Manual. All ABEND codes and message numbers used (directly or indirectly) by a module are defined in the extended description of the HIPO chart for that module. The actual ABEND code or message number issued by a particular segment (e.g., a common subroutine) may, in some cases, be ascertainable only in realtime execution. Therefore, not all PDL segments will reference a specific ABEND code or message number.

For ease of use, the following PDL charts have been arranged in alphabetical order.

Intentionally Blank

Figure 3-1. DOMICEXT

```
              DCMICEXT MAIN SEGMENT
01                INCLUDE DPPICINF
01                EXTA NUMERIC /* VALUE FOR 1PPS EXT'
01                EXTB NUMERIC  /*VALUE FOR FAIL SELECT EXT*/
01                /* THIS MODULE IS ENTERED IN PLACE OF THE OS EXT FLIH*/
01                IF EXTA IS ON THEN
02                  IF 'TIME SYNC OPTION IS SELECTED IN A JOB STEP' THEN
03                      'POST TIME SECTION'
02                  ENDIF
01                ENDIF
01                IF EXTB IS ON THEN
02                  IF 'PROBE FUNCTION IS RUNNING' THEN
03                      'POST PROBE'
02                  ENDIF
01                ENDIF
01                'PASS CTL TO OS/VS EXT FLIH'
              ENDSEGMENT DOMICEXT
```

Figure 3-2 (1 of 2). DOMIRBT

```
    DOMIRBT ENTERED VIA HARDWARE IPL
    SET UP TEMP ADDR TRANS TABLE
    TURN CN DAT BOX
    WAIT STATE CODE 'F' IF DSCB FOR F/R CATA SET NOT THERE
    IF CONSCLE ADDR FURNISHED
01    WRITE OUT MESSAGE THAT RESTART HAS OCCURED
    ENDIF
    UNTIL TC ENC OF REAL STORAGE
01    IF READ ADDR NOT FIRST PART BOOT
01    IF RED NOT SECCND PART BOOT
01    IF NOT ADDR WORK AREA
02      READ BLOCK CF REAL STORAGE
02      IF FIRST BLCCK REAL
03        SAVE PC NEW/MC NEW
03        PUT WAIT STATE PSWS AT PC AND MC
02      ENDIF
02      GET AMT READ
02      IF BLOCKS NCT 2K MULT
03        QUIT-DROP DEAD WAIT
02      ENDIF
02      GET NEXT READ ADDR
01    ELSE
02      UP REAC ADDR 2K
01    ENDIF
    ENDDO
    ALL REAL STORAGE IN--GET FULL CTL REGS
    READ PROTECT KEYS
    UNTIL END CF REAL STORAGE
```

Figure 3-2. (2 of 2). DOMIRBT

```
01     SET KEY IN REAL STORAGE
     ENDDO
     TO RENIP TO ADJLST UCB S
     IF CRITICAL DISK MISSING
01     CODE 3 WAIT STATE
     ENDIF
     DO COPY FOR PAGING CATA SETS
     UNTIL ALL JOBQ,SWADS PROC
01     COPY AN EXTENT OF JOBQ,SWADS,SYSWADS
01     SET NEXT EXTENT PTR
     ENDDO
     IF OS/VS CLOCK CMP SUPPORT
01     IF CLCCK SET OR
01     IF CLOCK NCT SET
02       ADJUST CLCCK CMP RESTORE VAL
02       WHILE TQE S REMAIN
03         ADJUST TOX IN TQE
03         GET NEXT TQE
02       ENDCO
01     ENDIF
01     SET CPU TIMER AND CLCCK CMP
     ENDIF
     RESTORE PC ANC MC NEW PSWS
     RETURN TO DCMIRWT AS THOUGH RESTART JUST WRITTEN
     BGNSEG COPY SEGMENT-COPY A DATA SET
01     IF ACTUAL DSET IMAGE THERE
02       UNTIL END OF INPUT ON F/R DSET
03         RELOC REAC CHAN PROG
03         GET AMT TREAD IN
03         BUILD OUTPUT CHAN PROG
03         GET DEVICE ADDR
03         WRITE IT CUT
03         UP INPUT DISK ADDR
03         IF END CYL ON INPUT
04           ADJUST HH AND CC
03         ENDIF
02       ENDDO
01     ENDIF
     ENDSEG COPY
```

Figure 3-3. DOMIRCMN

```
            DCMIRCMN MAIN SEGMENT
01              /* THIS SEGMENT IS PATCHED ON A CYCLIC BASIS TO
02                MONITOR THE EMS SYSTEM——IT CONVERSES WITH THE
02                BACKUP CPU VIA DIRECT CTL  */
01              INCLUDE D1MRCMN
01              SAVESTAT UNDEFINED   /*SAVELOCS FOR CTRS*/
01              LISTARYS CHARACTER /* LIST OF ARRAYS TO CHECK*/
01              FIRSTENT  BOOLEAN   /*INITIAL ENTRY SW */
01              GOODVAL  BOOLEAN
01              IF FIRSTENT=YES THEN
02                FIRSTENT=NO
02                'TURN OFF LTFREQ AND LTSELT'
02                'ALLOCATE SAVESTAT'
02                GOODVAL=TRUE
01              ENCIF
01              DO 'FOR ALL VALUES IN LISTARYS' WHILE GOODVAL=TRUE
02                IF 'LISTARYS VALUE=SAVED VALUE IN SAVESTAT' THEN
03                    IF 'LISTARYS VALUE NE ZERO' THEN
04                         GOODVAL=FALSE
03                    ENDIF
02                ELSE
03                    SAVESTAT(X)=LISTARYS(X)
02                ENDIF
01              ENCDO
01              IF GOODVAL=TRUE THEN
02                'SET BTDC TO NEW CONFIG'
02                'TURN ON LTCNL AND LTRDY'
01              ELSE
02                'SET BTDC TO 'F''  /* INDICATES FAILOVER REQ*/
01              ENDIF
            ENDSEGMENT DCMIRCMN
```

Figure 3-4. DOMIRCPY

```
   DOMIRCPY MAIN SEGMENT
   DCMIRCPY IS BRANCHED TO BY DOMIRFLV TC MAKE MULTIPLE
   COPIES OF THE FAILOVER/RESTART CATA SET.
   BUILD CCNTROL BLCCKS FOR ALL CPPFAIL DC CARCS
01       COPY ALL OF F/R DATA SET EXCEPT FIRST TRACK
01       WRITE BOOTSTRAP AND PROTECT KEYS ON EACH TRACK
01       READ R1 AND R2 FROM DISK TRACK ZERO AND COPY
01       READ IPL 1 ANC IPL2
01       WRITE OUT IPL1 AND IPL2 TO EACH DPPFAILX
   CLOSE CCBS,RELEASE BUFFERS
   ENDSEGMENT DCMIRCPY
```

Figure 3-5. DOMIRFLV

```
    DOMIRFLV SYNCHRONIZES WITH THE SLAVE(MASTER) PARTITION
    (IF ANY) ANC PREPARES TO WRITE THE RESTART
    ABEND NOT A REAL TIME JOB
    ABEND IF ANOTHER WTFAILDS GOING
    IF 2 PART OP
01    ABEND IF CANT LOCATE OTHER PARTITION
01    GET TCB ADDR CTHER PARTITION
01    ENQ CN MASTER JOB JOB NAME
01    IF E GOT RESOURCE
02      WAIT FOR OTHER PARTITION
01    ELSE
02      POST OTHER PARTITION
01    ENDIF
    ENDIF
    IF IS SLAVE PARTITICN
01    WAIT FOR MASTER TO WRITE RESTART
    ENDIF
```

Figure 3-6. DOMIRFL2

```
     DOMIRFL2 PREPARES TO WRITE THE RESTART
     IF MASTER PART.
01     IF RESTART IS ALLCWED
02       IF DPPFAIL DDCARD FOUND
03         IF NOT SYSRES (IPL VOL)
03         IF VALID DEVICE
04           LOAD RESTART BOOTSTRAP
04           LCAD RENIP
04           LOAD RESTART WRITE
04           PUT EXTENT INFO FOR JOBQUEUE AND SWADS IN BCOT HDR
04           IF SWADS NOT SWA
04           ENDIF
04           IF TWO PARTITION OPERATION
05             IF SWADS NOT SWA
06               PUT EXTENT INFC FCR SWADS OF SLAVE PARTITION IN BOOT HDR
05             ENDIF
04           ENDIF
04           GET I/O WORK AREA
04           PUT CONSOLE ADDR IN BCT HDR
04           FIX BCOT,RNIP,WORK AREA, AND RESTART WRITE IN REAL STORAGE
04           LOOP TIL I/O STOPS
04           GO TO DOMIRWT TO WRITE RESTART
04           IF ON RESTARTED CPU
05             RESET NR,CUB,BSY CN ALL UCBS
04           ENDIF
04           UNFIX ITEMS FIXED
03         ELSE
04           SET INVALID DDCARD FETURN
03         ENDIF
02       ELSE
03         SET NO CDCARD RETURN
02       ENDIF
02       IF RESTART WRITE OK
03         GC TO DCMIRCPY TC COPY DPPFAIL
02       ENDIF
01     ELSE
02       MAKE LOCK AS IF RESTART WENT OK
01     ENDIF
01     ISSUE APPROPRIATE MESSAGES
01     POST OTHER PARTITION IF EXIST
     ENDIF
     RETURN TO CALLER
```

Figure 3-7. DOMIRINT

```
     DOMIRINT MAIN SEGMENT
     INCLUDE DPPICINF
     /* THIS MODULE IS LINKED TO BY SYSINIT IF FAILOVER/RESTART
     AND OR EXT INT. HANDLING WAS SYSGENED */
     IF NOT SLAVE PART
01     ENQ ON NAME REPRESENTING FAILOVER/EXT INT GO WITH THIS JOB
01     IF ENQ OK
02       IF CAN HAVE EXT
03         IF EXT FLIH NOT PREV INIT
04           SET UP SUBSTITUTE EXT FLIH
03         ENDIF
03         IF EXT TIME STND EXT INT IN SYS
04           ATTACH TIME DRIFTER CORRECTOR TASK - DPPDRIFT
03         ENDIF
02       ENDIF
01     ELSE
02       SET NO RESTART/EXT INT THIS JOB
01     ENDIF
     ELSE SLAVE PART
01     SET EXT/INT RST OFF
     ENDIF
     ENDSEGMENT DOMIRINT
```

Figure 3-8. DOMIRNIP

```
   DCMIRNIP MAIN SEGMENT
   FORMAT BOOTSTRAP WORK AREA FOR CONFLICT ELEMENTS
   WHILE NOT AT END UCB LOOK UP
01   IF NOT FILLER ENTRY
02     IF DIRECT ACCESS DEVICE
03         READ VOLUME LABEL
03         IF UNABLE TC READ LABEL
04           IF DEVICE CNLINE BEFORE
04           IF REAL VCLSER
05             CREATE CONFLICT ELM
04           ENDIF
03         ELSE WAS ABLE TO READ VOL LABEL
04           IF DEVICE WAS OFFLINE OR
04           IF VCLSER IS DIFFERENT
05             CREATE CFLICT ELM
04           ENDIF
03         ENDIF
02     ENDIF
01   ENDIF
   BGNWHILE
01   GET NEXT UCB ADDR
   ENDDO
   REMOVE RQE S FRCM LOGICAL CHANNEL QUEUES AND CHAIN TOGETHER
   WHILE CFLICT ELMS STILL TO BE CHECKED
01   IF A NOW VCLERS EXISTS IN THIS ELM
02     SET UP TO SCAN CTHER CFLICT ELMS
02     STRTSRCH ELMS TC CHECK
02     EXITIF NOW OF ONE EQLS CLD OF ANOTHER AND
02     EXITIF SAME DEVTYP
03       INTERCHANGE UCB S
03       RESWAP VOLUME RELATED INFO
03       UNCHAIN CFLICT ELM
02     ORELSE
03       NEXT TO CCMPARE AGAINST
02     ENDLOOP
03       GET NEXT TO CHECK FOR VALID NOW VOLSER
02     ENDSRCH
01   ELSE
02     GET NEXT TO CHECK FOR VALID NOW VOLSER
01   ENDIF
   ENDDO
   PUT RQE S BACK CN CCRRECT LCH
   SET RETURN CODE IF CRITICAL DISK MISSING
   ENDSEGMENT DCMIRNIP
```

Figure 3-9. DOMIRPRB

```
            DCMIRPRB MAIN SEGMENT
01              /* THIS SEGMENT RUNS ON A CYCLIC BASIS IN THE
02                CFFLINE CPU--IT TESTS THE ONLINE CPU VIA DIRECT CTL */
01              INCLUDE D1MRCMN
01              INCLUDE DPPICINF
01              ECB(2) UNDEFINED  /*(1) IS TIME,(2) IS EXT INT */
01              GOODY BOOLEAN
01              'SAVE INFO IN DOMICINF FOR EXT INT'
01              GOODY=TRUE
01              'SET ECB(1) POSTED'
01              DO INDEFINITELY
02                'WAIT FOR ECE(1) OR ECB(2) TO BE POTED'
02                IF 'ECB(1) IS POSTED' THEN
03                    'SET ECB(1) NOT POSTED'
03                    'READ BTDC'
03                    IF BTDC IS ZERO THEN
04                         GOODY=TRUE
03                    ELSE
04                         IF BTDC EQ 15 THEN
05                              GOODY=FALSE
04                         ELSE
05                              IF 'BTDC IS THE SAM AS LAST TIME' THEN
06                                   GOODY=FALSE
05                              ENDIF
04                         ENDIF
03                    ENDIF
02                ENDIF
02                IF 'ECB(2) IS POSTED' THEN
03                    GOODY=FALSE
02                ENDIF
02                'SET LTRDY ON'
02                'SET STIMER FOR NEXT ENTRY'
02                IF GOODY=FALSE THEN
03                    IF 'ECB(2) IS NOT POSTED' THEN
04                         'SET LTFREQ ON'
03                    ELSE
04                         'SET LTFREQ AND LTSELT ON'
04                         'SET BT2914 AND DELAY'
04                         'FORCE IPL OF FAIL/RST CATA SET' /*EXIT*/
03                    EDNIF
02                ENDIF
01              ENDDO
            ENDSEGMENT DOMIRPRB
```

Figure 10. DOMIRWT

```
DOMIRWT IS BRANCHED TO BY COMIRFL2 TO WRITE THE FAILOVER/RST
DATA SET. IT RETURNS AFTER DOING SO.
SAVE GEN REGS IN BOOT HDR
SAVE CTL REGS IN BOOT HDR
INCLUDE DOMIRSIO
SAVE CLK CMP FOR RESTART READ
PUT RESUME PSW IN BCOT HEADER
DUMP ALL OF REAL STORAGE EXCEPT FIRST 2K WORK AREA AND BOOT(
3 2K BLOCKS IN ALL.
COPY ACTIVE ENTRIES FROM PAGING DATA SET TO DDPFAIL
COPY SYS1.SYSJOBQU AND SYS1.SYSWADS
COPY SWADS FOR MASTER PARTITICN
COPY SWADS FOR SLAVE PARTITION IF EXIST
WRITE PROTECT KEYS AND BOOTSTRAP
WRITE TRACK ZERC IPL RECORDS
RETRUN TO DIMIRFLV
```

Figure 3-11. DOMISVC1

```
            DOMISVC1 MAIN SEGMENT
01              /* TYPE 1 SVC ROUTER */
01               INCLUDE DOMISVCO
01               IF 'SUBSVC VALID' THEN
02                   CASENTRY SUBSVC
03                         CASE 1
04                             CALL    /* NO RET DISABLE SVC*/
03                         CASE 2
04                             CALL          /* NO RET CHAIN SC */
03                  ENDCASE
02               ELSE
03                  'ABEND USING SVCNO'
02               ENDIF
01          ENCSEGMENT DCMISVC1
```

Figure 3-12. DOMISVC2

```
            DCMISVC2 MAIN SEGMENT
01              /* TYPE 2 SVC ROUTER */
01               INCLUDE DOMISVCO
01               IF 'SUBSVC VALID' THEN
02                   CASENTRY SUBSVC
03                         CASE 1
04                             CALL      /* PATCH-NO RETURN*/
03                  ENCCASE
02               ELSE
03                  'ABEND USING SVCNO'
02               ENDIF
01          ENDSEGMENT DCMISVC2
```

Figure 3-13. DOMISVC4

```
            DOMISVC4 MAIN SEGMENT
01              /* TYPE 4 SVC ROUTER */
01              INCLUDE DOMISVCO
01              IF 'SUBSVC VALID' THEN
02                  CASENTRY SUBSVC
03                      CASE 1
04                         CALL DOMIRFLV   /* NO RETURN*/
03                  ENDCASE
02              ELSE
03                  'ABEND USING SVCNO'
02              ENDIF
01            ENDSEGMENT DOMISVC4
```

Figure 3-14. DOMXSTG1

```
            DCMXSTG1 MAIN SEGMENT
01            VALID_SW BOOLEAN
01            CONFG                   FILE
01            SOFTOPT                 FILE
01            OUTPUT                  FILE
01            SYSPRINT                FILE
01            HIEARCHY_CHAIN UNDEFINED /*PTR AND LIST OF PAR/CHILD*/
01            PARM_FIELD  CHARACTER /* PARM FILED FRCM EXEC CARD*/
01            'OBTAIN PARM_FIELD'
01            DO 'FOR EACH CPU IN PARM_FIELD'
02               'OPEN CONFG,SOFTOPT, AND OUTPUT FILES'
02               VALID_SW=TRUE
02               'SET HIERARCHY_CHAIN TO 1 STATED CPNFG MEMBER'
02               DO WHILE VALID_SW =TRUE UNTIL 'HIEARCHY_CHAIN AT END
03                  'READ CONFIGH MACRC FROM CURRNET MEMBER'
03                  IF 'CCNGIGH MACROS VALID' THEN
04                     'INSTALL CHILDREN IN HIEARCHY_CJAIN'
04                     'COPY CCNFG MEMBER TO OUTPUT FILE'
04                     'SET FOR NEXT MEMBER%IF ANY'
03                  ELSE
04                     VALID_SW=FALSE
03                  ENCIF
02               ENDDC
02               IF VALID_SW=TRUE THEN
03                  'CGPY SOPFTOPT MEMBER TO CUTPUT'
02               ELSE
03                  PUT SYSPRINT 'INVALID CONFIGH'
02               ENDIF
02               CLCSE OUTPUT
02               IF VALID_SW=TRUE THEN
03                  CALL 'ASSEMBER'
02               ENDIF
01            ENDDO
            ENDSEGMENT DOMXSTG1
```

Figure 3-15. DPCALCF1

```
     DPCALCF1-INCLUDED SEGMENT          DPPCALCF
 01    TIME  BIN                        GET OS TIME AND DATE
 01    STORE TIME AND CATE IN PARAMETER AREA
     ENDSEGMENT
```

Figure 3-16. DPCTIME1

```
    DPCTIME1-INCLUDED SEGMENT          DPPCTIME
01    IF CURRENT TIME GREATER THAN 24 HOURS
02      SET CCRRECTION FACTOR TO MINUS 24 HOURS
02      LINK DPPCUPCF                  UPCATE TIME
01    ELSE
02      CALCULATE TIME IN HCURS-MIN-SEC-DECISECONDS
02      STORE TIME IN HOURS-MIN-SEC-DECISECONDS IN TIME ARRAY
02      STCRE TIME IN BINARY DECISECONDS IN TIME ARRAY
01    ENDIF
    ENDSEGMENT
```

Figure 3-17. DPCTIME2

```
    DPCTIME2-INCLUDED SEGMENT          DPPCTIME
01    MESSAGE 'TIME CHANGED'
01    LINK   DPPCALCF                  CALCULATE NEW CORRECTION FACTOR
01    LINK   DPPCUPCF                  UPCATE TIME
    ENDSEGMENT
```

Figure 3-18. DPCTSVC1

```
        DPCTSVC1-INCLUDED SEGMENT      RET OPTICN REQUEST
01        READ TOD CLCCK
01        CALCULATE CURRENT TIME
01        RETURN TIME IN REG 0 ANC ADDRESS OF TIME ARRAY IN REG 1
        ENDSEGMENT
```

Figure 3-19. DPCTSVC2

```
            DPCTSVC2-INCLUDED SEGMENT    ADD OPTICN REQUEST
01            CBGET PTQE CCNTROL BLOCK
01            INITIALIZE PTQE WITH DATA IN PTIME INPUT PARAMETER-PTIMEL
          IF PRCBL IS LESS THAN 8 BYTES LONG AND
          IF PROBL IS NCT TO BE FREEC BY PTIME
01          MOVE PROBL INTC THE PTCE
          ENDIF
01            IF TCC SPECIFIED
02              IF START TIME LESS THAN CURRENT TIME
03                SET START TIME AHEAD BY 1 CAY
02              ENDIF
01            ELSE
02              IF REL SPECIFIED
03                ACD CURRENT TIME TO START TIME
02              ELSE                     ASSUME ADJ SPECIFIED
03                WHILE START TIME LESS THAN CURRENT TIME
04                  ADD INTERVAL TO START TIME
03                ENDDO
02              ENDIF
01            ENDIF
01            IF STOP TIME SPECIFIED
02              IF REL SPECIFIED
03                ADD CURRENT TIME TO STOP TIML
02              ELSE
03                IF ACJ SPECIFIED
04                  WHILE STOP TIME LFSS THAN CURRENT TIME
05                    ADD INTERVAL TO STOP TIME
04                  ENDDO
03                ENDIF
02              ENDIF
02              WHILE STOP TIME LESS THAN START TIME
03                ADD 24 HCUR VALUE TO STOP TIME
02              ENDDO
02              CALCULATE THE NUMBER OF INTERVALS
01            ELSE
02              IF ZERO CCUNTS SPECIFIEC
03                SET INFINITE PTIME FLAG
02              ENDIF
01            ENDIF
01            SAVE CCUNT VALUE
01            LCCK CN TIME ARRAY
01            CHAIN PTQE TO PTQE CHAIN
01            UNLOCK TIME ARRAY
            ENCSEGMENT
```

Figure 3-20. DPCTSVC3

```
        DPCTSVC3-INCLUDED SEGMENT     MOD SPECIFIED
01        LOCK ON TIME ARRAY
01        WHILE MORE PTQE'S ON CHAIN  DO FOR ALL PTQE'S
02          IF THIS IS A PTQE TO BE MODIFIED
03            CHAIN REMOVE PTQE FROM PTQE CHAIN
03            RESET PTQE WITH INFORMATION IN PTIME INPUT PARAMETER-PTIMEL
02        IF PROBL IS LESS THAN 8 BYTES LONG AND
02        IF PROBL IS NOT TO BE FREED BY PTIME
03          MOVE  PROBL INTO THE PTQE
02        ENDIF
03            IF TOD SPECIFIED
04              IF START TIME LESS THAN CURRENT TIME
05                SET START TIME AHEAD BY 1 DAY
04              ENDIF
03            ELSE
04              IF REL SPECIFIED
05                ADD CURRENT TIME TO START TIME
04              ELSE                  ASSUME ADJ SPECIFIED
05                WHILE START TIME LESS THAN CURRENT TIME
06                  ADD INTERVAL TO START TIME
05                ENDDO
04              ENDIF
03            ENDIF
03            IF STOP TIME SPECIFIED
04              IF REL SPECIFIED
05                ADD CURRENT TIME TO STOP TIME
04              ELSE
05                IF ADJ SPECIFIED
06                  WHILE STOP TIME LESS THAN CURRENT TIME
07                    ADD INTERVAL TO STOP TIME
06                  ENDDO
05                ENDIF
04              ENDIF
04              WHILE STOP TIME LESS THAN START TIME
05                ADD 24 HOUR VALUE TO STOP TIME
04              ENDDO
04              CALCULATE THE NUMBER OF INTERVALS
03            ELSE
04              IF ZERO COUNTS SPECIFIED
05                SET INFINITE PTIME FLAG
04              ENDIF
03            ENDIF
03            SAVE COUNT VALUE
03            CHAIN ADD THIS PTQE TO PTQE CHAIN
02          ENDIF
01        ENDDO
01        UNLOCK TIME ARRAY
         ENDSEGMENT
```

Figure 3-21. DPCTSVC4

```
         DPCTSVC4-INCLUDED SEGMENT      DEL SPECIFIED
01         LOCK ON TIME ARRAY
01         WHILE MORE PTQE'S ON CHAIN DO FOR ALL PTQE'S
02           IF THIS IS A PTQE TO BE DELETED
03             SET PURGE FLAG
02           ENDIF
01         ENDDO
01         POST DPPCPTIM                 TO REMOVE PTQE FROM PTQE CHAIN
01         UNLOCK TIME ARRAY
         ENDSEGMENT
```

Figure 3-22. DPCUPCF1

```
     DPCUPCF1-INCLUDED SEGMENT           DPPCUPCF
01     CONVERT INPUT CORRECTION FACTOR TO TOD COUNTER UNITS
02       IF   TIME IS FAST THEN
03         ADD CORRECTION FACTOR TO CURRENT CONVERSION FACTOR
02       ELSE
03         SUBTRACT CORRECTION FACTOR FROM CURRENT CONVERSION FACTOR
02       ENDIF
01 *           NOTE: THE CONVERSION FACTOR IS SUBTRACTED FROM TAE
01 *                 TOD COUNTER VALUE TO OBTAIN THE TIME
02     STORE NEW CONVERSION FACTOR INTO TIME ARRAY
01   ENDSEGMENT
```

Figure 3-23. DPCUPCF2

```
DPCUPCF2-INCLUDED SEGMENT           DPPCUPCF
 READ   TOD COUNTER VALUE
 SUBTRACT CORRECTION FACTOR FROM TOD COUNTER VALUE
 CONVERT TIME DIFFERENCE TO 10 MILLISECOND UNITS
 STORE  BINARY TIME VALUE INTO TIME ARRAY
 CONVERT BINARY TIME VALUE INTO HOURS-MINUTES-SECONDS-DECISECONDS
 STORE  HOURS-MINUTES-SECONDS-DECISECONDS INTO TIME ARRAY
ENDSEGMENT
```

Figure 3-24. DPCUPCF3

```
DPCUPCF3-INCLUDED SEGMENT         CPPCUPCF
 LOAD    INPUT JULIAN DATE
 STORE   JULIAN CATE IN TIME ARRAY
 STORE   JULIAN DAY OF THE YEAR INTO TIME ARRAY
 CONVERT JULIAN DAY OF THE YEAR TO BINARY
 CALCULATE MONTH-CAY-YEAR FRCM JULIAN DATE
 STORE   MONTH-CAY-YEAR INTC TIME ARRAY
 CONVERT MONTH-DAY-YEAR INTO EBCDIC
STORE  EBCCIC  MCNTH-DAY-YEAR INTO TIME ARRAY
ENDSEGMENT
```

Figure 3-25. DPCUPCF4

```
     DPCUPCF4-INCLUDED SEGMENT          DPPCUPCF
 01    GET ADCRESS CF FIRST PTQE ON CHAIN
 01    IF INPUT CHANGE VALUE IS NEGATIVE
 02      IF  INPUT CHANGE VALUE IS EQUAL TO 24 HOURS THEN ITS MIDNIGHT
 03        WHILE PTCE ADDRESS IS NOT ZERO
 04          RESET PTQE TIME BY 24 HOURS
 03        ENDDO
 02      ELSE
 03        WHILE PTCE ADDRESS IS NOT ZERO
 04          IF ORIGINAL START TIME LESS THAN CURRENT TIME OR
 04          IF ORIGINAL START TIME GREATER THAN TIME OF NEXT PATCH
 05            USE THE CURRENT TIME AS START TIME
 04          ELSE
 05            USE THE CRIGINAL START TIME
 04          ENDIF
 04          RESET NUMBER OF PATCH COUNTS
 04          RESET NEXT PATCH TIME
 03        ENDDO
 02      ENDIF
 02      ENDIF
 01    ENDIF
     ENDSEGMENT
```

Figure 3-26. DPIDBAS1

```
   DPIDBAS1-INCLUDED SEGMENT          READ INITIALIZATION DATA
    OPEN DATA BASE DCB
    BLDL TC GET DIRECTORY ENTRY FOR @INIT ARRAY
    READ @INIT ITEM RECORD
    CALCULATE CATA BASE TABLE SIZES FRCM INFCRMATICN IN ITEM RECORD
    GETMAIN PROTECTED CORE FOR DATA BASE TABLES
01    GETMAIN USER CORE FOR CBLCB AND CBCC TABLES
    UNTIL ALL @INIT CATA RECORDS HAVE BEEN READ IN
01    READ  CATA RECORD INTO GETMAINEC AREA
    ENDDO
   ENDSEGMENT DPIDBAS1
```

Figure 3-27. DPIDBAS2

```
DPIDBAS2-INCLUCEC SEGMENT          BUILD TABLE HEADERS
 OPEN    COMPOSITE ITEMS DCB AND DATA BASE DCB
 CALCULATE PAGE BCUNDARY TABLE SIZE
 GFTMAIN PROTECTED CORE FCR PBT
 BUILD   PBT
 INITIALIZE DATA BASE TABLE HEADERS
ENDSEGMENT DPICBAS2
```

Figure 3-28. DPIDBAS3

```
    DPIDBAS3-INCLUDED SEGMENT          CHAIN CATA BASE TABLES TOGETHER
     GETMAIN CCRE FCR VS RESIDENT ARRAY DATA
     UNTIL ALL ARRAYS HAVE BEEN PROCESSED
 01    IF IT'S NOT A DUMMY ARRAY
 02      IF IT'S A VS RESIDENT ARRAY
 03        IF INITIAL CATA REQUIRED
 04          BLDL FCR ARRAY DIRECTORY ENTRY
 04          UNTIL ALL CATA RECCRDS HAVE BEEN READ
 05            READ ARRAY DATA RECORD
 04          ENDDO
 03        ENCIF
 02      ELSE
 03        IF CCB FOR CA RESIDENT ARRAY DDNAME HAS NOT BEEN OPENED
 04            DEFLCCK FOR THIS DCB ACDRESS
 03          CPEN  DCB FCR DDNAME
 02        ENDIF
 01      ENDIF
       ENDIF
     ENDDO
    ENDSEGMENT DPICBAS3
```

Figure 3-29 (1 of 2). DPINIT01

```
    DPINIT01 - INCLUDED SEGMENT - PARAM KEYWORD PROCESSOR
    WAS A PARAM= KWD PREV PROCESSED ?
    IF NOT - PROCESS KWD
01    TURN ENDING FLAGS OFF
01    TURN ON PARAM= KWC PROCESSED FLG
01    IF PARAM BEGINS WITH LEFT PARENTHESIS
02      UNTIL *:LOOP - COUNT QUOTES TO GET # SUBPARAMS
03        IF INCR COUNTER
03        ENDIF
02      ENDDO
02      IF COUNT EVEN - PROPERLY BALANCED
03        BUILD A PRCBL
03        UNTIL LOOP TILL END OF PARAM PROCESSING
04          IF FCRMAT IS CORRECT, IDENTIFY DATA
05            IF CATA TYPE IS X'  '
06              UNTIL LCOK FOR ENDING QUOTE
06              ENDDC
06              SAVE ADDR OF ENDING QUOTE
06              CALC LNTH OF X CATA
06              IF FIELC CCNTAINEC ALL VALID HEX DATA
07                TRANSLATE THE HEX DATA
07                GET CORE FOR PARM
07                STORE  PARM ADDR IN PROBL
07                PUT LNTH IN PROBL
06              ELSE
07                WRITE ERROR MSG
06              ENDIF
05            ELSE
06              IF DATA TYPE IS F'  '
07                GET STORAGE FOR DATA
07                PUT PARM ADDR IN PROBL
07                PUT CCRE LNTH IN PROBL
07                IF DATA HAS PLUS SIGN
08                  PCINT PAST SIGN CHARACTER
07                ENDIF
07                IF DATA HAS MINUS SIGN
08                  PCINT PAST SIGN CHARACTER
08                  TURN ON FULLWORD NEGATIVE FLG
07                ENDIF
07                UNTIL SCAN FOR ENDING QUOTE
07                ENDDO
07                CCNVERT INPUT TO BINARY
07                IF FULLWORD NEGATIVE FLAG ON
08                  CCMPLIMENT  CATA
07                ENDIF
```

Figure 3-29 (2 of 2). DPINIT01

```
07                PUT BIN VALUE IN PARM
07                IF *:NOPDL
07                ELSE
08                  IF END OF PARAM PROCESSING
08                  ELSE
08                  ENDIF
07                ENDIF
06              ELSE
07                IF CATA TYPE C
08                  UNTIL LOOK FOR ENDING QUOTE
08                  ENDDO
08                  SAVE ENDING QUOTE ACDR
08                  CALC DATA LNTH
08                  GET STORAGE FOR DATA
08                  PUT PARM ADDR IN PROBL
08                  PUT LNTH IN PROBL
08                  MOVE DATA TO PARM AREA
08                  WRITE ERROR MSG
07                ENDIF
06              ENDIF
05            ENCIF
04          ELSE
05            WRITE ERROR MSG
04          ENDIF
03        ENDDO
02      ELSE
03        WRITE ERROR MSG
02      ENDIF
01    ELSE
02      WRITE ERROR MSG
01    ENDIF
    ELSE
01    WRITE ERROR MSG
    ENDIF
    IF NC ERRORS - UPCATE PTRS
01    IF END OF STATEMENT REACHEC
02      TURN CN END FLG
01    ELSE
02      IF MORE OPERANDS ARE EXPECTED
03        PCINT TO START OF NEXT FIELD
02      ELSE
03        WRITE ERROR MSG
02      ENDIF
01    ENDIF
    ENDIF
```

Figure 3-30 (1 of 4). DPINIT02

```
   DPINIT02 - INCLUDED SEGMENT - PATCH CARD PROCESSOR
   IF THIS IS A PRE RESTART PATCH BLOCK
01   TURN ON PRE-RESTART PTCH FLAG
   ENDIF
   WHILE DO UNTIL ENC FLG OR ERR FLG
01   IF KWD  FOUND
02     IF KEWORD = EP
03       IF EP = KWD NOT ALREACY PROCESSED
04         TURN ON EP KWD  PROCESSED FLG
04         IF EP NAME LE 8 CHAR
05           MOVE NAME TO SUPL
05           IF END OF OPERAND PROCESSING
06             TURN CN END FLAG
05           ELSE
06             INCR PTR TO START OF NEXT KEYWORD
05           ENDIF
04         ELSE
05           WRITE ERRCR MSG
04         ENDIF
03       ELSE
04         WRITE ERROR MSG
03       ENDIF
02     ELSE
03       IF KWD = TASK
04         IF TASK = KWD NOT PREVIOUSLY PROCESSED
05           TURN CN TASK= KWD PROCESSED FLG
05           IF TASK NAME LE 8 CHARACTERS
06             MOVE NAME TO SUPL
06             IF END OF OPERANDS
07               TURN CN END FLAG
06             ELSE
07               POINT TO FIRST CHAR OF NEXT KWD
06             ENDIF
05           ELSE
06             WRITE ERROR MSG
05           ENDIF
04         ELSE
05           WRITE ERRCR MSG
04         ENDIF
03       ELSE
04         IF KWD = QL
05           IF QL KWD NOT PREVIOUSLY PROCESSED
06             TURN CN CL= KWD PROCESSED FLG
06             IF QL LE VALUE CF 100
07               IF CATA VALID
08                 IF ID VALUE LE 255           DR180
09                   PUT QL IN SUPL
09                   IF END OF DATA
10                     INDICATE END OF OPERANDS
09                   ELSE
10                     INCR PTR TO START OF NXT KWD
09                   ENDIF
```

Figure 3-30 (2 of 4). DPINIT02

```
08            ELSE DR180
09              WRITE ERROR MESSAGE          DR180
08            ENCIF DR180
07          ELSE
08            WRITE ERROR MSG
07          ENDIF
06        ELSE
07          WRITE ERROR MSG
06        ENDIF
05      ELSE
06        WRITE ERROR MSG
05      ENDIF
04    ELSE
05      IF KWD=ID
06        IF ID = KWD NOT PREVIOUSLY PROCESSED
07          TURN CN ID= KWD PROCESSED FLG
07          IF ID VALUE LE 255
08            IF CATA VALID
09              IF ID VALUE LE 255       DR180
10                SAVE THE ID
10                IF NO MCRE  OPERANDS
11                  TURN ON END FLG
10                ELSE
11                  INCR TO FIRST CHAR CF NXT CPND
10                ENDIF
09              ELSE DR180
10                WRITE ERROR MESSAGE          DR180
09              ENDIF DR180
08            ELSE
09              WRITE ERROR MSG
08            ENDIF
07          ELSE
08            WRITE ERROR MSG
07          ENDIF
06        ELSE
07          WRITE RRROR MSG
06        ENDIF
05      ELSE
06        IF KEYWORD = PRTY
07          IF PRTY= KWD NOT PREVIOUSLY PROCES
08            TURN CN PRTY= KWD PROC FLG
08            IF CODED 'TASKNAME,N'
09              IF TASKNAME LE 8 CHAR
10                IF DELIMETER WAS A COMMA
11                  MOVE PRTY REFERENCE NAM TO SUPL
11                  STRTSRCH FIND END OF FIELD
11                  EXITIF END FOUND
12                    IF FIRST CHAR NOT RPAREN
13                      WRITE ERROR MSG
```

Figure 3-30 (3 of 4). DPINIT02

```
12          ELSE
13            IF DATA VALID
14              IF PRTY VALUE LE 255 DR180
15                PUT PRTY VALUE IN SUPL
15                IF NO MORE OPERANDS
16                  TURN ON END FLG
15                ELSE
16                  IF VALID DELIMETER
17                    POINT TO START OF NXT OPND
16                  ELSE
17                    WRITE ERROR MSG
16                  ENDIF
15                ENDIF
14              ELSE DR180
15                WRITE ERROR MESSAGE  DR180
14              ENDIF DR180
13            ELSE
14              WRITE ERROR MSG
13            ENDIF
12          ENDIF
11        ORELSE
12          INCR TO NEXT CHAR
11        ENDLOOP
12          WRITE ERROR MSG
11        ENDSRCH
10      ELSE
11        WRITE ERROR MSG
10      ENDIF
09    ELSE
10      WRITE ERROR MSG
09    ENDIF
08  ELSE
09    IF PRTY CODED JOBSTEP-N
10      IF PRTY LE 3 CHAR
11        IF DATA VALID
12          PUT REL PRTY VALUE IN SUPL
12          IF END OF PROCESSING
13            TURN ON END FLG
12          ELSE
13            INCR TO START OF NXT OPND
12          ENDIF
11        ELSE
12          WRITE ERROR MSG
11        ENDIF
10      ELSE
11        WRITE ERROR MSG
10      ENDIF
```

Figure 3-30 (4 of 4). DPINIT02

```
09                   ELSE
10                     WRITE ERROR MSG
09                   ENDIF
08                 ENDIF
07               ELSE
08                 WRITE ERROR MSG
07               ENDIF
06             ELSE
07               IF KEYWORD=PARAM
08                 COPY DPINIT01
07               ELSE
08                 WRITE ERROR MSG
07               ENDIF
06             ENDIF
05           ENDIF
04         ENDIF
03       ENDIF
02     ENDIF
01   ELSE
02     WRITE ERROR MSG
01   ENDIF
   BGNWHILE
   ENDDO
   IF NO ERRORS - CHECK PROBL
01   IF NO PROBL - GET ONE
01   ENDIF
01   PUT ID IN PROBL
   ENDIF
   IF NO ERRORS WERE DETECTED
01   IF EP= WAS NOT SPECIFIED
02     WRITE ERR MSG
01   ENDIF
   ENDIF
   CLEAR PATCH FLAGS
```

Figure 3-31 (1 of 2). DPINIT03

```
     DPINIT03 - INCLUDED SEGMENT - CCNTINUATICN CARD PROCESSOR
     IF COLUMNS 1-15 ARE BLANK
01     IF PROCESSING IS NOT IN PARAM FIELD
02       IF VALID CCNTINUATION
03         LCCK FOR FIRST BLANK CR QUOTE
03         IF END CF OPNDS BEFORE END DATA COL
04           IF DELIMITER IS A BLANK
05             IF CHAR BEFORE DELIM IS A COMMA
06               TURN ON CONTINUATION FLAG
05             ELSE
06               CHECK FCR BLANK IN COL 72
06               IF COLUMN 72 IS BLANK
07                 TURN ON NO OPERANDS AND CCNTINUATION FLAGS
06               ELSE
07                 TURN OFF NO OPERANDS ANC CONTINUATION FLAGS
06               ENDIF
05             ENDIF
04           ELSE
05             BREAK OUT BLANKS (DO DPINIT06)
04           ENDIF
03         ELSE
04           IF CCNTINUATION EXPECTED
05             TURN ON CCNTINUATION EXPECTED FLG
04           ELSE
05             TURN CCNTINUATICN FLG OFF
04           ENDIF
03         ENDIF
02       ELSE
03         IF COLUMN 72 IS NON-BLANK
04           TURN OFF NO OPERANDS AND CONTINUATICN FLAGS
03         ENDIF
03         WRITE ERROR MSG
02       ENDIF
01     ELSE
02       FIND THE ENDING QUOTE FOR PARAM
02       IF DELIM FOUND
03         BREAK OUT BLANKS (DO DPINIT06)
02       ELSE
03         IF NO CCNTINUATION EXPECTED
04           WRITE ERROR MSG
04           TURN CONT FLG OFF
03         ELSE
04           TURN CCNTINUATION FLAG CN
03         ENDIF
02       ENDIF
01     ENDIF
```

Figure 3-31 (2 of 2). DPINIT03

```
01    IF MAX OPERANDS NCT EXCEECED
02      MOVE OPNDS TO WA
01    ELSE
02      IF NO CCNTINUATION EXPECTED
03        TURN OFF FLAGS
02      ENDIF
02      WRITE ERROR MSG
01    ENDIF
    ELSE
01    IF NO CCNTINUATION EXPECTED
02      TURN OFF CONTINUATION AND NO MORE OPERANDS FLAGS
01    ENDIF
01    WRITE ERRCR MSG
    ENDIF
```

Figure 3-32 (1 of 5). DPINIT04

```
      DPINIT04 - INCLUDED SEGMENT - CONTRCL CARD PROCESSOR
      IF ANY MORE CCNTINUATIONS
01      IF NC MORE ERRORS HAVE BEEN DETECTED
02        IF THIS IS A PATCH CARD
03          COPY DPINIT02 - PATCH CARD PROCESSOR
02        ELSE
03          IF THIS IS A WAIT CARD
04            IF NAME LE 8 CHAR.
05              IF ANY INITCB'S EXIST ON CHAIN
06                STRTSRCH SEARCH TO END OF CHAIN
07                  LCCK FOR PATCH CTL BLCK
06                EXITIF BLOCK FOUND WITH PROPER LABEL
07                  IF NAMED BLOCK IS A PATCH BLOCK
08                    GET A CB ANC ADD IT TO CHAIN
08                    GET ECB ADDR
08                    PUT ECB ADCR IN WAIT BLK
07                  ELSE
08                    WRITE ERROR MSG
07                  ENDIF
06                CRELSE
07                  UPDATE TO NEXT BLOCK
06                ENDLOOP
07                  WRITE ERROR MSG
06                ENDSRCH
05              ELSE
06                WRITE ERROR MSG
05              ENDIF
04            ELSE
05              WRITE ERROR MSG
04            ENDIF
03          ELSE
04            IF THIS IS A RESTART STATEMENT
05              IF NO WRITE RESTART PREVIOUSLY FOUND
06                TURN CN RESTART WRITE FLAG
06                GET AN INITCB AND CHAIN IT
06                TURN CN WRITE BLOCK FLAG
06                UNTIL ALL OPERANDS PROCESSED,CO
07                  IF THIS IS A WRITE RESTART REQUEST
08                    TURN ON WRITE RESTART FLAG
07                  ELSE
08                    IF THIS IS A RESTART NOWRITE
09                      LEAVE  WRITE RESTART FLAG IN ITS PRESENT STATE
08                    ELSE
09                      IF THIS IS A CANCEL FLAG
10                        TURN ON THE CANCEL FLAG
09                      ELSE
10                        IF THIS IS NOCANCEL RECUEST
11                          LEAVE CANCEL FLAG IN ITS PRESENT STATE
```

Figure 3-32 (2 of 5). DPINIT04

```
10                    ELSE
11                      IF THIS IS A PROBE REQUEST
12                        TURN ON PROBE FLAG
11                      ELSE
12                        IF NOPROBE REQUESTED
13                          LEAVE PROBE FLAG IN ITS PRESENT STATE
12                        ELSE
13                          IF CCNT MOITOR REQUEST
14                            TURN CMON FLG ON
13                          ELSE
14                            IF NOCMON REQUESTED
15                              LEAVE THE CMON FLAG IN ITS PRESENT STATE
14                            ELSE
15                              WRITE ERROR MESSAGE
14                            ENDIF
13                          ENDIF
12                        ENDIF
11                      ENDIF
10                    ENDIF
09                  ENDIF
08                ENDIF
07              ENDIF
07              IF NO ERRORS CETECTED
08                IF DELIMETER FCUND
08                ELSE
09                  WRITE ERRCR MESSAGE
08                ENDIF
07              ELSE
08                SET CONDITICN TO EXIT DO LOOP
07              ENDIF
06            ENDDO
05          ELSE
06            WRITE ERROR MSG
05          ENDIF
04        ELSE
05          IF THIS IS A MASTER CARD
06            IF TWC PARTITION IS ALLOWED
07              IF NO PREV MAST OR SLAVE CARD READ
08                IF KEYWORD IS SLAVE=
09                  IF JOBNAM LE 8 CHAR
10                    MOVE NAME TO MAIN BLOCK
10                    TURN ON MASTER FLAG
09                  ELSE
10                    WRITE ERROR MSG
09                  ENDIF
08                ELSE
09                  WRITE ERROR MSG
08                ENCIF
07              ELSE
08                WRITE ERROR MSG
07              ENDIF
```

Figure 3-32 (3 of 5). DPINIT04

```
06              ELSE
07                WRITE ERROR MESSAGE
06              ENDIF
05            ELSE
06              IF THIS IS SLAVE CARD
07                IF TWC PARTITION IS ALLOWED
08                  IF NO MAST OR SLAV PREVIOUSLY READ
09                    IF KEYWORD IS MASTER=
10                      IF JOBNAME LE  8 CHAR
11                        MOVE JN TO MAIN BLOCK
11                        TURN ON  SLAVE FLAG
10                      ELSE
11                        WRITE ERROR MSG
10                      ENDIF
09                    ELSE
10                      WRITE ERROR MSG
09                    ENDIF
08                  ELSE
09                    WRITE ERROR MSG
08                  ENDIF
07                ELSE
08                  WRITE ERROR MESSAGE
07                ENDIF
06              ELSE
07                IF THIS IS A CBGET CARD
08                  IF CPND LE  2 CHAR
09                    IF DATA VALID
10                      PUT # 2K BLKS CBGET CORE IN MAINBLOK
09                    ELSE
10                      WRITE ERROR MSG
09                    ENDIF
08                  ELSE
09                    WRITE ERROR MSG
08                  ENDIF
07                ELSE
08                  IF THIS IS A GETWA CARD
09                    IF SUBLIST STARTS WITH LPAREN
10                      UNTIL END OF LIST - OR
10                      UNTIL ERROR FOUND
11                        IF DELIMITER FOUND
12                          IF # PARAMS LE MAX
13                            IF DELIMITER IS BLANK
14                              IF LAST COL NOT RPAREN
15                                WRITE ERROR MSG
14                              ENDIF
13                            ENDIF
```

Figure 3-32 (4 of 5). DPINIT04

```
13              IF NO ERRORS DETECTED
14                IF DATA VALID
15                  IF #BLKS
16                    IF # INVALID
17                      WRITE ERROR MSG
16                    ENDIF
15                  ELSE
16                    IF SIZE INVALID
17                      WRITE ERROR MSG
16                    ENDIF
16                    IF SIZE GT 2K
17                      IF VALUE NOT 2K MULTIPLE
18                        WRITE ERROR MESSAGE
17                      ENDIF
16                    ENDIF
15                  ENDIF
15                  IF NON ZERO      DR5063
16                    PUT VALUE IN LIST
15                  ELSE DR5063
16                    WRITE ERROR MSG   DR5063
15                  ENDIF DR5063
14                ELSE
15                  WRITE ERROR MSG
14                ENDIF
13              ENDIF
12            ELSE
13              WRITE ERROR MSG
12            ENDIF
11          ELSE
12            WRITE ERROR MSG
11          ENDIF
11          END OF SUBLIST ?
10        ENDDO
09      ELSE
10        WRITE ERROR MSG
09      ENDIF
09      IF NO ERRORS DETECTED
10        SORT TABLE ENTRIES BY SIZE
09      IF THIS IS TCB CARD
10        IF OPERAND LE  THREE CHARACTERS
11          IF DATA VALID
12            PUT # TCB'S IN MAINBLOK
11          ELSE
12            WRITE ERROR MSG
11          ENDIF
```

Figure 3-32 (5 of 5). DPINIT04

```
10                      ELSE
11                        WRITE ERROR MSG
10                      ENDIF
09                    ELSE
10                      IF THIS IS AN ABEND CARD
11                        GET A CB AND CHAIN IT
11                        ENDIF
10                      ELSE
11                        IF THIS IS A DBREF CARD
12                          IF CBREF NOT WANTED
13                            IF CBREF REQUESTED
14                              LEAVE THE DBREINIT FLAG ON
13                            ELSE
14                              WRITE ERROR MESSAGE
13                            ENDIF
12                          ELSE
13                            GET MAINBLOK ADDR
13                            TURN OFF DBREINIT FLAG
12                          ENDIF
11                        ELSE
12                          WRITE ERROR MSG
11                        ENDIF
10                      ENDIF
09                    ENDIF
08                  ENDIF
07                ENDIF
06              ENDIF
05            ENDIF
04          ENDIF
03        ENDIF
02      ENDIF
01    ENDIF
    ENDIF
```

Figure 3-33. DPINIT05

```
   DPINIT05 - INCLUDED SEGMENT - BUILD WAIT LIST ROUTINE
   IF A CONT WAS EXPECTED BUT NOT RECVD
01   ISSUE ERROR MESSAGE
   ENDIF
   IF ANY CONTROL STATEMENT ERRORS FOUND
01   ISSUE ERRCR MESSAGE
01   ABEND - USER 34
   ENDIF
   IF THIS IS NOT A SLAVE PARTITION
01   IF NO PATCH BLOCK EXISTS
02     ABEND TFE JOB - USER 40
01   ELSE
   IF A WRITE RESTART BLOCK EXISTS
01   UNTIL LOOP COUNTING PATCH BLOCKS
02     IF THIS A PATCH BLCCK
03       INCR COUNT
02     ENDIF
01   ENDDC STOP WHEN WRITE BLOCK FOUND
01   IF PATCH BLOCKS PRECEED THE WRITE
02     BUILD A WAIT LIST
01   ENDIF
   ENDIF
   IF WRITE RESTART NOT LAST BLK ON CHAIN
01   UNTIL COUNT PATCH BLKS WITH PARM
02     IF THIS A PATCH BLOCK
03       IF PARAM WAS CODED
04         COUNT PATCH BLKS WITH PARAM
03       ENDIF
02     ENDIF
02     GET PTR TC NEXT BLK
01   ENDCO
01   IF PATCH BLOCKS EXIST
02     UNTIL ENO OF CHAIN
03       IF THIS A PATCH BLOCK
04         IF PATCH HAS A PARAM
05           GET ECB ADDR
05           PUT ECB ACDR IN ECB LIST
05           INCR TO NEXT LIST ENTRY
04         ENDIF
03       ENDIF
03       GET NEXT BLOCK ADDRESS
02     ENDDO
01   ENDIF
   ENDIF
   RETURN TO CALLER
```

Figure 3-34. DPINIT06

```
   DPINIT06 - INCLUDED SEGMENT - RTNE TO HANDLE BLANKS IN PARAM
   BGNSEG
01   UNTIL VALIC CELIMETER FCUND OR
01   UNTIL DELIMETER AND ERROR OR          DR183
01   UNTIL END OF CATA COLUMN REACFED
02     IF BLANKS ARE NOT DELIMETERS
03       IF QUOTE FCUND BEFORE END OF CATA
04         MAKE BLANK A DELIMITER
03       ELSE
04         IF NC CCNTINUATION - ERROR
05           WRITE ERROR MESSAGE
05           TURN CONT FLAG OFF
04         ELSE
05           TURN ON CONTINUATION FLAG
04         ENDIF
03       ENDIF
02     ELSE
03       IF DELIM FOUND
04         IF THE DELIMITER IS A QUOTE
05           MAKE BLANKS A DELIMITER
04         ELSE
05           IF LAST CATA COL IS A COMMA
06             TURN CN CONTINUATION AND  DELIMETER FLAGS
05           ELSE
06             IF CCNTINUATICN COLUMN IS NON BLANK
07               TURN FLGS ON
07               TURN CONTINUATION, NO OPERANDS, AND DELIMETER FLAGS ON
06             ELSE
07               MAKE BLANK A DELIMITER
07               TURN OFF CONTINUATION AND NO OPERANDS FLAGS
06             ENDIF
05           ENDIF
04         ENDIF
03       ELSE
04         IF LAST DATA COL IS CCMMA
05           TURN  CONTINUATICN FLG ON
04         ENDIF
04         IF CCNTINUATION COL IS NON BLANK
05           TURN CONTINUATICN FLG ON
04         ENDIF
03       ENDIF
02     ENDIF
01   ENDDO
   ENDSEG DPINIT06
```

Figure 3-35. DPINIT08

```
   DPINIT08 - INCLUDED SEGMENT - INITCB BUILD AND CHAIN ROUTINE
   BGNSEG
01   GET CORE FOR CONTROL BLOCK
01   CLEAR CORE GOTTEN FOR CTL BLK
01   GET CHAIN ORIGIN
01   WHILE LOOP TILL END OF CHAIN
02     UPDATE CURRENT PTR
01   BGNWHILE
02     GET NEXT PTR
01   ENDDO
01   ADD NEW BLOCK TO END OF CHAIN
   ENDSEG DPINIT08
```

Figure 3-36. DPINIT09

```
   DPINIT09 - INCLUDED SEGMENT - ERROR MESSAGE WRITER ROUTINE
   BGNSEG
01   TURN ON CONTROL STATEMENT ERROR FLAG
01   GET MESSAGE ADDR IN REG 0
01   IF OUTPUT DCB OPEN
02     PUT ERROR MSG TO WRITER
01   ENDIF
   ENDSEG DPINIT09
```

Figure 3-37. DPINIT1

```
   DPINIT1 - INCLUDED SEGMENT - INITIALIZE TMCT AND GETWA CORE
   CALCULATE AMOUNT OF SPACE REQUIRED FOR TMCT
   GETMAIN FROM SUBPOOL 253 FOR TMCT
   CLEAR  TMCT CORE
   GET MY LIMIT PRIORITY
   GET DIFF FOR DPPTPMCNS PRIORITY
   CALCULATE DPPTPMCN'S LIMIT PRTY
   PUT DPPTPMCN'S LIM PRTY IN TMCT
   PUT GFMB ADDR IN TMCT
   GET THE NUMBER OF GETWA SIZES
   PUT # GWA SIZES IN TMCT
   PUT XCVT ADDR IN TMCT
   CALCULATE THE AMOUNT OF PROTECTED STORAGE REQUIRED FOR GETWA CONTROL
   BLOCKS:
   *       (NUMBER SIZESXGFCBLNTH)+(TOT NUMB BLOCKSXGFBELNTH)=CORE SIZE
   IF MINIMUM SIZE NOT SPECIFIED APAR24
01   ABEND JOB STEP - USER 46   APAR24
   ENDIF APAR24
   INITIALIZE GETWA CONTROL BLOCKS:
   *                  - GFCB = GETWA-FREWA CONTROL BLOCKS
   *                  - GFBE = GETWA-FREEWA BLOCK ENTRY
   *                  - GFMB = GETWA-FREEWA MAIN BLOCK
   UNTIL GETWA CONTROL BLOCKS INITIALIZED
01   PUT SIZE OF GETWA BLOCKS IN GFMB
01   POINT THE GFMB TO ITS GFCB
01   INITIALIZE ID FIELD OF GFMB
01   INCREMENT ID FOR NEXT GFMB
01   POINT THE GFCB TO ITS GFMB
01   MAKE PREV GFCB POINT TO THIS GFCB
01   *      THE FIRST ONE WILL BE POINTED TO
01   *      BY THE TMCT
01   UPDATE TO NEXT GFMB
01   POINT TO FIRST GFBE
01   POINT GFCB TO FIRST FREE GFBE
01   TURN ON INITIAL ALLOCATION FLAG
01   CALC ADDR OF NEXT GFCB
01   WHILE THERE ARE GFBE'S TO BE CHAINED - DO
02     GET ADDR OF NEXT GFBE
02     POINT TO NEXT FREE GFBE
02     UPDATE BASE TO NEXT GFBE
01   ENDDO
01   ZERO NEXT PTR IN LAST GFBE
   ENDDO
   ZERO NEXT PTR IN LAST GFCB
   CALCULATE TOTAL CORE REQUIRED FOR GETWA - DO A GETMAIN FOR THE CORE
   * THEN ALLOCATE IT TO THE GFCB'S
```

Figure 3-38. DPINIT11

```
   DPINIT11 - INCLUDED ROUTINE - POST CODE ERROR RCUTINE
   BGNSEG
01   GET ECB ACCR CLEAR HI-ORDER BYTE
01   GET ADDR OF INITCB WITH BAD POST CODE DR5C87
01   ISSUE THE ERRCR MESSAGE WITH ECB CONTENTS
01   IF THIS IS A PRE-RESTART PATCH
02     TURN ON POST ERROR FLAG
01   ENDIF
   ENDSEG DPINIT11      *:          RETURN TO CALLER
```

Figure 3-39. DPINIT12

```
 BGNSEG  DPINIT12 - PROBE INTERFACE
  CIRB   CREATE IRB
  SCHEDULE IRB UNCER TIME TASK (DPPCTIME) TO SUPPRESS
*                TIMING UNTIL PRCBE CCMPLETES
  PATCH  PROBE FUNCTION  (DCMIRPWT)
  WAIT   TILL  IT CCMPLETES
  POST   IRB TO EXIT
 ENDSEG  DPINIT12
```

Figure 3-40. DPINIT13

```
    BGNSEG  DPINIT13 - CONTINUOUS MONITOR INTERFACE
01    PATCH CONTINUOUS MONITOR (DCMIRCMN)
    ENDSEG  CPINIT13
```

Figure 3-41. DPINIT14

```
    BGNSEG  CPINIT14 TIME IRB ROUTINE
01    WAIT  FCR   PRCBE TO CCMPLETE
    ENDSEG  DPINIT14
```

Figure 3-42. DPINIT2

```
   DPINIT2 - INCLUDEC SEGMENT - INITIALIZE CBGET CORE
   CALCULATE TFE AMOUNT OF CCNTRCL BLOCK (PRCTECTED) CORE TO GET
   IF CBGET CORE WAS REQUESTED AT RUN TIME
01   GET THE NUMBER OF 2K BLOCKS REQUESTEC
01   CALC AMCUNT OF CBGET CORE REQUIRED
   ELSE
01   GET # TCB'S REQUESTED
01   GET TFE TCBX LENGTH
01   CALC AMT OF CORE REQ'D FOR TCBX'S
01   ROUND TO 2K
01   ADD SAFETY FACTOR TO CBGET CORE REQUIREC
   ENDIF
   GETMAIN FOR CBGET CORE FROM SUBPOOL 253
   INITIALIZE CBGET CORE AND CCNTRCL BLOCKS
   INITIALIZE A PROTECTED STORAGE CONTROL BLOCK (PSCB) AT THE START
   *  OF CBGET CORE AND ALLOCATE ALL CORE TO IT
   INITIALIZE # FREE 32 BYTE BLOCKS
   GET DUMMY PSCB ADDR IN SCVT
   CHAIN NEW PSCB TO SCVT
   MAKE SCVT LAST USEC PTR PT TO NEW
```

Figure 3-43. DPINIT3

```
   DPINIT3 - INCLUDED SEGMENT
   LOAD SYSTEM MONITOR - DPPTSMON
   LOAD ETXR RCUTINE - DPPTETXR
   PUT INITIAL # TCB'S IN TMCT
   PUT # FREE TCB'S IN TMCT
   DO A GETMAIN FOR A TCBX - INITIALIZE IT AND CHAIN IT TO THE          *
   *   CURRENT TCB WHICH WILL BECCME THE TCB FOR DPPTSMCN               *
   LOAD PATCH MCNITOR - DPPTPMON
   LOAD PMCN STAE RCUTINE    PRF#166
   LOAD PURGEWQ ROUTINE
   LOAD TRANSWA ROUTINE
   UNTIL ACVANCE TCB'S GOTTEN AND INITIALIZED - DO
01   CREATE ANC CHAIN ADVANCE TCB'S AND TCBX'S
01   IF CBGET CORE NOT AVAILABLE
02     ABEND - USER 33
01   ENDIF
01   INITIALIZE GOTTEN TCBX, CHAIN IT ON FREE Q , AND USE POST TO CHAIN
01   * IT TO TCB
   ENDDO
   COPY DPINIT5 - TWO PARTITION SYNC ROUTINE
   LOAD DPPXDEFL - DEFLOCK ROUTINE
   LCAD CPPTGWFW - GET/FREEWA BR SUBROUTINE
   PUT GETWA BR SUBRTNE ADDR IN THE SCVT
   PUT XCVT ADDRESS IN GETWA SUBROUTINE
   LOAD CPPXLCCK - LCCK ROUTINE
   PUT OS CEFAULT SVC'S IN DUPLICATE CATA SET ROUTINE ADDRESSES
   IF DDS WAS GENERATED
   LOAD CCS ROUTINES
   PUT DDS ROUTINES ADCRESSES IN SCVT
   LOAD CPPXDRCX - CATA RECORD DUMMY ROUTINE
   LINK TO DATA BASE INITIALIZATICN
   IF A WRITE RESTART CARD WAS PROCESSED
01   ZERO REG1  FOR  DPPMINIT
   ELSE
01   NON-ZERO REG 1  TC SHOW NO WRT-RST
   ENDIF
   LINK TC MSG HANDLER INITIALIZATION
   LINK TO TIME MANAGEMENT INITIALIZATION
   LINK TO DB LCGGING INITILIZATION
```

Figure 3-44 (1 of 2). DPINIT5

```
     DPINIT5 - INCLUDED SEGMENT - TWO PARTITION SYNC ROUTINE
     IF BYPASS SYNC CODE
     ELSE
01     IF THIS IS THE MASTER PARTITION
02       ENQ MASTER-MASTER JOBNAME
01     ELSE
02       ENQ MASTER-MASTER JOBNAME
01     ENDIF
01     IF THIS IS NOT A RESTART OF SLAVE PARTN
02       IF THIS IS THE MASTER PARTITION
03         ENQ MYNAME - OTHERNAME (MASTER- SLAVE)
02       ELSE
03         ENQ OTHERNAME - MYNAME (MASTER- SLAVE)
02       ENDIF
02       IF RET CODE GT 0 - OTHER PTN STARTED
03         STRTSRCH FIND OTHER
03         EXITIF JOBNAME FOUND
03         EXITIF WITH NON-ZERO TCB KEY
04           PUT HIS XCVT ADDR IN MY XCVT
04           POST HIM WITH MY XCVT ADDR
03         ENDLOOP
04           ABEND - USER 36
03         ENDSRCH
02       ELSE
03         GET THE ADDR OF MY 2ND XCVT FIELD
03         PUT IT IN ECB LIST TO USE AS ECB
03         ISSUE TIMER WAIT
03         ISSUE SYNC MSG
03         WAIT ON ONE OF THE ECB'S
03         IF TIMER EXPIRED
04           CLEAR ECB
04           BRANCH BACK AND REISSUE TIMER WAIT
03         ELSE
04           IF I AM THE MASTER PARTITION
05             DEQ MYNAME - OTHERNAME (MASTER- SLAVE)
04           ELSE
05             DEQ OTHERNAME - MYNAME (MASTER- SLAVE)
04           ENDIF
04           STRTSRCH
05             FIND THE OTHER JOBNAME
04           EXITIF OTHER PTN FOUND
04           ENDLOOP
05             ABEND - USER 36
04           ENDSRCH
04           GET HIS XCVT ADDR
03         ENDIF
02       ENDIF
02       IF THIS IS THE MASTER PARTITION
03         ENQ JOBNAME MASTER-MASTER
02       ENDIF
```

Figure 3-44 (2 of 2). DPINIT5

```
02       PUT HIS TCB ADDR IN MY SCVT
02       PUT HIS SCVT ADDR IN MY SCVT
02       PUT HIS LO BOUNDARY IN MY SCVT
02       PUT HIS HI BOUNDARY IN MYSCVT
01     ELSE
02       THIS IS A SLAVE PARTITION BEING RE-STARTED AFTER A FAILURE
02       STRTSRCH SEARCH TCB CHAIN
02       EXITIF MASTER JOBNAME FOUND IN A PARTN
02       EXITIF WITH A NZERO TCB KEY
02       ENDLOOP
02       ENDSRCH
02       IF MASTER IS NOT CURRENTLY ABENDING
03         IF THERE IS NO SLAVE CURRENTLY RUNNING
04           TURN ON TWO PARTITION AND SLAVE RE-SYNC FLAGS
04           PUT MY SCVT ADDR IN HIS SCVT
04           PUT MY XCVT ADDR IN HIS XCVT
04           PUT MY TCB IN HIS SCVT
04           PUT HIS TCB IN MY SCVT
04           PUT HIS SCVT ADDR IN MY SCVT
04           PUT HIS XCVT ADDR IN MY XCVT
04           PUT MY PTN LO BNDRY IN HIS SCVT
04           PUT MY PTN HI BNDRY IN HIS SCVT
04           PUT HIS PTN LO BNDRY IN MY SCVT
04           PUT HIS PTN HI BNDRY IN MY SCVT
03         ELSE
04           ABEND - USER 44
03         ENDIF
02       ELSE
03         ABEND - USER 43
02       ENDIF
01     ENDIF
     ENDIF
```

Figure 3-45. DPPCALCF

```
     DPPCALCF-MAIN SEGMENT                 CALCULATE CONVERSION FACTOR
01     INITIALIZE  DPPCALCF WORK AREA
01     COPY  CPCALCF1                      CALCULATE EXTERNAL TIME
01     PTIME RET                           GET CURRENT TIME
01     SUBTRACT     TIME FROM EXTERNAL TIME
01     STORE TIME DIFFERENCE IN PARAMETER AREA
     RETURN
```

Figure 3-46. DPPCPTIM

```
    DPPCPTIM-MAIN SEGMENT              ISSUE PATCHES
01    INITIALIZE WCRK AREAS
01    SETPSW TO GET PRCTECT KEY 0
01    POST  DPPITIMI TO LET HIM KNOW THAT 'IM READY
01    UNTIL JOBSTEP TASK ABENDS        FOREVER AND EVER AND...
02      WAIT UNTIL POSTED BY CPPCTSVC CR CPPCTIME
02      LCCK ON TIME ARRAY
02      IF  POSTEC BY CPPCTSVC         REMCVE FTQC
03        WHILE MORE PTQE'S ON CHAIN
04          IF THIS PTCE IS TO BE DELETED
05            IF PTQE TO BE DEPATCH
06              CEPATCH
05            ENDIF
05            IF PTQE TO BE POSTED
06              POST ECB
05            ENDIF
05            IF PROBL TO BE FREED
06              FREEMAIN PROBL
05            ENDIF
05            CHAIN REMOVE PTQE FRCM PTQE CHAIN
05            CBFREE FREE PTQE CCNTROL BLOCK
04          ENDIF
03        ENDDO
02      ENCIF
02      IF POSTED BY DPPCTIME
02      UNTIL ALL PTCE'S IN THIS TIME INTERVAL HAVE BEEN SERVICED
03        CALCULATE NEXT PTQE TIME
03        IF THIS WILL BE THE LAST PATCH
04          SET FLAG TO FREE PROBL DURING PATCH
03        ENDIF
03        PATCH   THIS PTQE
03        IF THIS WAS LAST PATCH FOR THIS PTCE OR THE PATCH WAS BAD
04          MESSAGE  SPECIFYING BAD PATCH RETURN CODE
04          IF DEPATCH SPECIFIED
05            CEPATCH
04          ENDIF
04          IF PTQE TO BE PCSTED
05            PCST ECB
04          ENDIF
04          IF PROBL TO BE FREED
05            FREEMAIN PROBL
04          ENDIF
04          CHAIN REMOVE PTQE FRCM PTQE CHAIN
04          CBFREE FREE PTQE CCNTROL BLOCK
03        ELSE
04          CHAIN REMOVE PTQE FROM PTQE CHAIN
04          CHAIN ADD PTQE TO CORRECT POSITICN ON CHAIN
03        ENDIF
02     ENCCO
02     ENDIF
02     UNLOCK TIME ARRAY
02    ENDDO
01  RETURN
```

Figure 3-47. DPPCTIME

```
    DPPCTIME-MAIN SEGMENT                   TIME UPCATE PROGRAM
01    INITIALIZE DPPCTIME WORK AREA
01    POST  CPPITIME                        LET HIM KNCW I'M READY
01    UNTIL JOB STEP TASK CANCELED          FOREVER,ANC EVER,AND EVER ...
02      STIMER WAIT FOR SYSGENED TIME INTERVAL
02      READ TOD COUNTER VALUE
02      CALCULATE CURRENT TIME CF CAY
02      IF CURRENT TIME OF DAY IS INCONSISTENT WITH THE EXPECTED TIME
03        COPY CPCTIME2                     CORRECT TIME
02      ELSE
03        COPY DPCTIME1                     CALCULATE CURRENT TIME
02      ENDIF
02      IF  PTCE SHOULC BE SERVICED DURING THIS INTERVAL
03        POST CPPCPTIM
02      ENDIF
02      UPCATE FAILCVER COUNT IN TIME ARRAY
01    ENDCC
    RETURN
```

Figure 3-48. DPPCTSVC

```
    DPPCTSVC-MAIN SEGMENT                   PTIME REQUESTS
01    INITIALIZE WORK AREA
01    IF    IT'S A RECUEST FRCM THE SLAVE PARTITION
02      SET SLAVE FLAG
01    ENDIF
01    CASE ENTRY                            DETERMINE TYPE OF PTIME REQUESTS
02      COPY      DPCTSVC1                  CALCULATE CURRENT TIME
02      COPY      CPCTSVC2                  ADD PTCE TO PTQE CHAIN
02      COPY      CPCTSVC3                  MODIFY PTQE'S ON PTQE CHAIN
02      COPY      DPCTSVC4                  DELETE PTQE'S CN PTQE CHAIN
01    ENC CASE
    RETURN
```

Figure 3-49. DPPCUPCF

```
    DPPCUPCF-MAIN SEGMENT                   UPDATE CCNVERSICN FACTOR
01    INITIALIZE  CPPCUPCF WORK AREA
01    LCCK  TIME ARRAY,TYPE=LOCK            EXCLUSIVE CCNTROL OF TIME ARRAY
01    COPY  DPCUPCF1                        UPDATE CONVERSION FACTOR
01    COPY  CPCUPCF2                        UPCATE TIME
01    COPY  DPCUPCF3                        UPCATE DATE
01    COPY  CPCUPCF4                        UPCATE PTQE'S
01    LOCK  TIME ARRAY,TYPE=UNLOCK          RELEASE CCNTROL OF TIME ARRAY
    RETURN
```

Figure 3-50 (1 of 3). DPPDARAY

```
    ***********************************************************************
    *                                                                     *
    *:*                              CPPCARAY                             *
    *:*                                                                   *
    *:*           CATA BASE SUPPORT ROUTINE FOR GETARRAY AND PUTARRAY.    *
    *:*                                                                   *
    *                                                                     *
    ***********************************************************************
    *                                                                     *
    *:*  FUNCTICN-                                                        *
    *:*           ENTERED VIA BALR FROM THE APPLICATION PROGRAM AS THE    *
    *:*           RESULT OF THE EXECUTION OF A GETARRAY OR PUTARRAY MACRO*
    *:*               THIS PROGRAM, WHEN ENTERED, DO CNE OF THE FOLLOWING*
    *:*                  1) MOVE INTO THE USER'S AREA, THE DATA FROM ONE CR *
    *:*                     MORE ARRAYS.                                  *
    *:*                  2) MOVE INTO THE USER'S AREA, THE ADDRESSES OF ONE *
    *:*                     OR MORE ARRAYS.                               *
    *:*                  3) MOVE INTO THE USER'S AREA, THE DEFINITION SPEC- *
    *:*                     IFICATIONS OF ONE OR MORE ARRAYS.             *
    *:*                  4) MOVE INTO THE DATA EASE, CNE CR MORE SPECIFIC *
    *:*                     ARRAYS OF CATA FROM THE USER'S AREA.          *
    *:*                                                                   *
    *                                                                     *
    ***********************************************************************
01     CCNVERT REQUEST OPTIONS TO INTERNAL FORMAT
01     SET RETURN CODE TC ZERC
01     ESTABLISH BEGINNING AND END OF REQUEST LIST
01     IF PUTARRAY CR PRCTECT=YES ANC CATA TO BE MOVED
02       SET DATA BASE LOCK
01     ENCIF
01     IF NAMES OR NUMBERS WERE SUPPLIED
02       DO UNTIL ALL ENTRIES PROCESSED
03         IF ARRAY NAMES SUPPLIED BY CALLER
04           SCAN PAGE ECUNDRY TABLE FOR NAME RANGE
04           SCAN SECCNDARY ARRAY LOCATOR FOR ARRAY NAME
04           CONVERT TABLE ENTRY TO ARRAY ID
03         ELSE    USER SUPPLIEC ARRAY NUMBER
04           IF USER SUPPLIED NUMBER NOT VALID
05             DC ERRCR
04           ENDIF
03         ENCIF
03         FIND ENTRY IN PRI. A.L.7. FOR THIS ARRAY
03         IF USEP REQUESTED ADDRESSES
04           MOVE' ARRAY ADDRESS, NO.OF BLKS AND BK SIZE
05                                  TO USER'S AREA
04         ELSE
```

Figure 3-50 (2 of 3). DPPDARAY

```
05          IF ARRAY SPECIFICATIONS REQUESTED
06            DC SPECMOVE
05          ELSE
06            IF DATA MOVE REQUESTEC
07              DO MOVEDATA
06            ENDIF
05          ENCIF
04        ENDIF
03      ENDDO
02    ELSE    USER SUPPLIED ADDRESSES
03      DO UNTIL ALL ENTRIES PROCESSED
04          ESTABLISH ADDRESSES IN DATA BASE AND USER'S AREA
04          IF GETARRAY EXECUTED
05            MOVE DATA FROM ARRAY TO USERS AREA
04        ELSE USER EXECUTED PUTARRAY
05          MOVE DATA FROM USER'S AREA TO DATA BASE
04        ENDIF
03      ENDDO
02    ENDIF
02    IF LOCK WAS SET
03      RELEASE LCCK CF DATA BASE
02    ENDIF
02    RETURN TO CALLER.
01 **
01 **
01 ** SEGMENT MCVECATA
01 **              VALICATE ALT ENTRY AND MOVE DATA
01 **
02    IF ARRAY IS CIRECT ACCESS RESIDENT
03      DC ERRCR      OPERATION INVALID
02    ELSE    IT IS VS RESIDENT
03      GET ARRAY BLOCK SIZE
03      IF BLOCKED ARRAY
04        MULTIPLY BY BLOCK COUNT
03      ENDIF
03      GET ADDRESS OF USER'S AREA
03      IF GETARRAY SPECIFIED
04        MOVE ENTIRE ARRAY TO USER'S AREA
03      ELSE  PUTARRAY WAS SPECIFIED
04        MOVE CATA FRCM USER'S AREA TO CATA BASE
03      ENDIF
02    ENDIF
02    END SEGMENT MOVECATA
01 **
01 **
01 ** SEGMENT ERRCR
01 **         SAVE CCUNT OF ERRORS ANC SET RETURN CODE
01 **
```

Figure 3-50 (3 of 3). DPPDARAY

```
02      IF FIRST ERROR
03        SAVE ADDR OF USER'S PARAMETERS
02      ENDIF
02      ADC 1 TO CCUNT CF ERRORS
02      SET RETURN CODE TO 4
02      END SEGMENT ERROR
01  **
01  **
01  **  SEGMENT SPECMCVE
01  **          USER REQUESTED SPEC'S FOR ITEMS IN ARRAY
01  **
02      ALLCCATE WORK SPACE
02      SET LOCK
02      FIND DCB FOR DBINIT DATA SET
02      POINT TO ITEM RECORD FOR THIS ARRAY
02      READ ALL RECORDS OF ITEM DEFINITIONS THIS ARRAY
02      RELEASE LCCK
02      RELEASE WORK SPACE
02      END SEGMENT SPECMOVE
```

Figure 3-51. DPPDBLOK

```
       DPPDBLCK-GETBLCCK/PUTBLOCK MAIN SEGMENT
01       UNTIL ALL ARRAYS HAVE BEEN PROCESSED
02         IF  NUMBERED ARRAY THEN
03           ARRAY ID  =  NUMBER
02         ELSE
03           FIND SALT PACE FOR THIS ARRAY NAME
03           FIND ARRAY NAME IN THIS SALT PAGE
03           CALCULATE ARRAY ID (ID=DISPLACEMENT INTO SALT TABLES / SALT SIZE)
02         ENDIF
03           FIND PALT
03           IF PROTECT REQUESTED
04             IF VS RESIDENT ARRAY
05               LCCK ON VS CATA BASE
04             ELSE
05               LCCK CN DA CATA BASE
04             ENDIF
03           ENDIF
03           UNTIL ALL BLCCKS OF THIS ARRAY HAVE BEEN MOVED
04             IF VS RESIDENT ARRAY
05               IF PUTBLOCK
06                 MOVE BLOCK INTO ARRAY
05               ELSE
06                 MOVE BLCCK INTO USER AREA
05               ENDIF
04             ELSE
05               IF PUTBLOCK
06                 WRITE BLOCK TO DA ARRAY
05               ELSE
06                 READ DATA INTO URER AREA
05               ENDIF
04             ENDIF
04            ENDDO
03           IF PROTECT REQUESTED
04             UNLOCK
03           ENDIF
02       ENDDC
01     ENDSEGMENT
```

Figure 3-52. DPPDBSIF

```
    DPPCBSIF-MAIN SEGMENT              CATA BASE TASK TO READ/WRITE CATA
   *                 FROM THE CATA BASE FOR THE SLAVE PARTITION
01     SETPSW  TO  CET PROTECT KEY ZERO TO STORE INTO SLAVE
01     BALR  TO CATA BASE ROUTINE REQUESTEC
   *         NOTE : CPPDSUB2 SETS UP THE REQUIRED REGISTERS THEN
   *                 PATCHES DPPCBSIF TO DO THE WORK
01     RETURN
    ENDSEGMENT
```

Figure 3-53. DPPDFREQ

```
    DPPDFREQ - TIME DRIVEN LOGGING
01     LOAD  ID FROM PARM LIST
01     IF    ID IS NZERO
01     LOAD  PUTLOG NO. LIST BASED ON THIS ID
01     PUTLOG
01     ELSE
02       BUILD UPCATED REFRESH ARRAY USING THE CURRENT LOG COPIES
02       PUTBLOCK  UPDATE REFRESH ARRAY
01     ENDIF
    ENDSEGMENT
```

Figure 3-54. DPPDGETL

```
    CPPDCETL - MAIN SEGMENT            GET LOG ROUTINE
01     IF    NUMBERED ARRAY
02       ARRAY ID = NUMBER
01     ELSE
02       FIND SALT PAGE FOR THIS ARRAY NAME
02       FIND ARRAY NAME IN THIS SALT PAGE
02       CALCULATE ARRAY ID (IC=DISPLACEMENT INTOSALT TABLES/ SALT SIZE)
01     ENDIF
01     FIND PALT
01     FIND CURRENT LOGHEADER
01     FIND LOG ARRAY PALT
01     IF NC REFERENCE TIME
02       USE CURRENT LCG TIME
01     ENDIF
01     BUILC GETBLOCK CATA LIST
01     SEARCH FOR REQUESTED BLOCK
01     GETBLOCK-READ IN REQUESTED LOG COPY
    ENDSEGMENT
```

Figure 3-55 (1 of 4). DPPDITEM

```
***********************************************************************
*                                                                     *
*:*                              DPPDITEM                             *
*:*                                                                   *
*:*            CATA BASE SUPPORT ROUTINE FOR GETITEM AND PUTITEM.     *
*:*                                                                   *
*                                                                     *
***********************************************************************
*                                                                     *
*:*  FUNCTICN-                                                        *
*:*            ENTERED VIA BALR FROM THE APPLICATION PROGRAM AS THE   *
*:*            RESULT OF THE EXECUTION OF A GETITEM OR PUTITEM MACRC. *
*:*                THIS PROGRAM, WHEN ENTERED, DO ONE OF THE FOLLOWING*
*:*               1) MOVE INTO THE USER'S AREA, THE CATA FROM CNE CR *
*:*                  MORE ITEMS.                                      *
*:*               2) MOVE INTO THE USER'S AREA, THE ADDRESSES OF CNE *
*:*                  OR MORE ITEMS.                                   *
*:*               3) MOVE INTO THE USER'S AREA, THE DEFINITION SPEC- *
*:*                  IFICATIONS OF ONE OR MORE ITEMS.                 *
*:*               4) MOVE INTO THE CATA BASE, CNE OR MORE SPECIFIC   *
*:*                  ITEMS OF DATA FROM THE USER'S AREA.              *
*:*                                                                   *
*                                                                     *
***********************************************************************
01    CONVERT INPUT OPTICNS TO INTERNAL FORMAT
01    SET RETURN CODE TO ZERO
01    IF PUTITEM SPECIFIED AND NOT DATA MOVE
02      SET RETURN CODE TO 8
01    ELSE
02      IF ITEM NAMES SPECIFIEC
03        DO NAMESOLV
03        IF ITEM SPECIFICATIONS REQUESTEC
04          DO MOVESPEC
03        ELSE
04          CO CCNVSPEC
04          IF ITEM ADDRESSES REQUESTEC
05            DO MOVEADDR
04          ELSE
05            IF DATA MOVE REQUESTED
06              DO MOVEDATA
05            ELSE
06              SET RETURN CODE TO 8
05            ENDIF
04          ENDIF
03        ENDIF
02      ELSE
03        IF ADDRESSES WERE SUPPLIEC
04          IF CATA WAS REQUESTED
05            DO MCVEDATA
04          ELSE
```

Figure 3-55 (2 of 4). DPPDITEM

```
05              SET RETURN CODE TO 8
04            ENDIF
03          ENDIF
02        ENDIF
01      ENDIF
01      RFTURN TO CALLER
   **
   **
   ** SEGMENT NAMESCLV
   **               RESOLVE ITEM NAMES TC ARRAY ID + DISPLACEMENT
   **
01      ESTABLISH NO. OF NAMES TO RESOLVE
01      ALLOCATE AND INITIALIZE WORK SPACE
01      SORT NAMES INTO COLLATING SEQUENCE
01      DO UNTIL ALL NAMES RESOLVED
02        READ NEXT BLCCK OF ARRAY @CIDS
03          IF NEXT NAME NOT IN THIS BLOCK
04            READ NEXT BLOCK
03          ELSE
04            MOVE ITEM SPEC. CATA TC WORK SPACE
04            STEP TO NEXT NAME
03          ENDIF
02      ENDDO
02      SORT NAMES AND CATA BACK TO ORIGINAL SEQUENCE
02      ENDSEG NAMESCLV
01 **
01 **
01 ** SEGMENT MOVESPEC
01 **               MOVE ITEM SPECIFICATIONS TO USER'S AREA
01 **
02      ESTABLISH ADCRESSES WHERE SPEC. CATA IS TO GO
03        DO UNTIL ALL ITEMS PRCCESSED
04          MOVE SPEC CATA
03        ENDDC
02      ENDSEG MCVESPEC
01 **
01 **
01 ** SEGMENT MOVEADDR
01 **               MOVE ITEM ADCRESSES TO USER'S AREA
01 **
02      ESTABLISH CCUNT CF ITEMS AND MOVE TO ADDRESSES
02      DO UNTIL ALL ITEMS PROCESSED
03        IF ADDRESS WAS NOT RESCLVED
04          SET RETURN CCDE TO 4
03        ELSE
04          MOVE ITEM ADDRESS TO USER'S AREA
03        ENDIF
02      ENDDC
```

Figure 3-55 (3 of 4). DPPDITEM

```
02    IF LIST CF NAMES WAS SPECIFIED
03      MOVE END OF LIST FLAG TO USER'S AREA
02    ENDIF
02    ENDSEGMENT MOVEADDR
01 **
01 **
01 ** SEGMENT MOVECATA
01 **                  MOVE ITEM DATA TO USERS AREA
01 **                  CR FROM USER'S AREA
01 **
02    IF CALL FRCM PUTITEM
03      SET TC MOVE TO DATA BASE
02    ELSE
03      SET TC MCVE TO USER'S AREA
02    ENDIF
02    IF PUTITEM OR PROTECT SPECIFIED
03      IF WORK SPACE NCT ALLOCATED
04        ALLOCATE AND FORMAT WORK SPACE
03      ENDIF
03      SET LCCK
02    ENDIF
02    IF CNLY CNE NAME SPECIFIED
03      IF THE ADDRESS WAS NOT RESOLVED
04        SET RETURN CODE TO 4
03      ELSF
04        IF ITEM LENGTH IS ZERO
05          SET RETURN CODE TO 4
04        ELSE
05          MCVE ONE ITEM OF DATA
04        ENCIF
03      ENDIF
02    ELSE
03      ESTABLISH AODR OF LIST OF ITEM ACDRESSES
03      IF USER DID NCT SPECIFY CATA INCREMENT-
04                        USE LENGHE AS DEFINED FOR EACH ITEM
03        DO UNTIL ALL ITEMS MCVED
04          IF CATA ADCRESS IS ZERO
05            SET RETURN CODE TO 4
04          ELSE
05            MOVE DATA FOR LENGTH OF ITEM
04          ENDIF
04          INCREMENT USER'S AREA ADCRESS BY LENGH OF ITEM
03        ENDDO
02      ELSE            USE THE INCREMENT SPECIFIEC BY USER
03        DO UNTIL ALL ITEMS MCVED
04          IF DATA ADDRESS IS ZERO
05            SET RETURN CODE TC 4
```

Figure 3-55 (4 of 4). DPPDITEM

```
04          ELSE
05            MOVE CATA FOR LENGTH OF ITEM
04          ENDIF
04          INCREMENT USER'S AREA ADDRESS BY INCR. SPECIFIED
03        ENDDO
02      ENDIF
01    ENDIF
01    IF LCCK WAS SET
02      RELEASE LCCK
01    ENDIF
01    END SEGMENT MOVECATA
   **
   **
   ** SECMENT CONVSPEC
   **             CCNVERT FROM ARRAY IC/CISPLACEMENT
   **             TO ADDRESS OF ITEM
   **
01    ESTABLISH COUNT AND ACCRESS CF ENTRIES
01    DO UNTIL ALL ENTRIES CONVERTED
02      IF ITEM NAME WAS RESOLVED
03        IF ITEM IS IN VS RESIDENT ARRAY
04          IF USER SPECIFIED BLOCK NUMBER
05            IF ITEM IS IN BLCCKED ARRAY
06              IF BLOCK NUMBER IS VALID
07                CALCULATE ITEM ACCRESS
06              ELSE
07                SET RETURN CODE TO 4
07                SET ITEM ADDRESS TO ZERO
06              ENDIF
05            ELSE    ITEM IS IN UNBLOCKED ARRAY
06              IF BLOCK NUMBER WAS SPECIFIED
07                SET RETURN CODE TO 4
07                SET ITEM ADDRESS TO ZERO
06              ELSE
07                CALCULATE ITEM ADDRESS
06              ENDIF
05            ENDIF
04          ELSE      THE USER DID NOT SPECIFY BLOCK NUMBER
05            CALCULATE ITEM ADDRESS
04          ENDIF
03        ELSE        IT IS FOR D.A. RESIDENT ARRAY
04          SET RETURN CODE TO4
04          SET ITEM ADDRESS TC ZERO
03        ENDIF
02      ELSE          THE ITEM NAME WAS NOT RESOLVED
03        SET RETURN CCCE TC 4
02      ENDIF
01    ENDDC
01    END SEGMENT CONVSPEC
```

Figure 3-56. DPPDPUTL

```
    DPPDPUTL-PUTLOG SUBROUTINE
01    UNTIL ALL ARRAYS HAVE BEEN PROCESSED
02      IF  NUMBERED ARRAY
03        ARRAY ID = NUMBER
02      ELSE
03        FIND SALT PAGE FOR THIS ARRAY NAME
03        FIND ARRAY NAME IN THIS SALT PAGE
03        CALCULATE ARRAY ID ( ID=DISPLACEMENT INTO SALT TABLES/ SALT SIZE)
02      ENDIF
02      FIND PALT
02      FIND CURRENT LCG HEADER
02      IF  LOGHDR SPECIFIED
03        FIND LCG ARRAY PALT
03        BUILD PUTBLCCK NUMBER LIST BASED ON LOG HEADER LOG COPY
03        PUTBLOCK
02      ELSE
03        IF BLKLIST SPECIFIED
04          FIND LCG ARRAY PALT
04          BUILD PUTBLOCK NUMBER LIST BASED ON CURRENT LOG COPY
04          PUTBLOCK
03        ENDIF
02      ENDIF
02      IF  NORMAL PUTLOG REQUEST
03        RESET CURRENT LOG COPY
03        IF LOG CCPY 1 THEN
04          SET FIRST LOG COPY TIME OF DAY
03        ENDIF
03        FIND LOG ARRAY PALT
03        BUILD PUTBLOCK NUMBER LIST BASED ON NEW LOG COPY
03        PUTBLOCK
03        IF THIS WAS THE LAST LCG COPY AND
03        IF A WRAP ARROUND ROUTINE WAS SPECIFIED
04          PATCH DEPENDENT TASK WRAP ARROUND ROUTINE
03        ENDIF
02      ENDIF
01    ENDDI
    ENDSEGMENT
```

Figure 3-57. DPPDRIFT

```
   DPPDRIFT MAIN SEGMENT
   DPPDRIFT IS THE TIME DRIFT CORRECTION MODULE
   UNTIL OUT OF INIT
01   WAIT FOR EXT INT AND GET TOD VALUE
   ENDDO
   IF S370/EMS IN SYSTEM
01   PATCH EMS TIME START UP TASK
   ELSE
01   LINK TO TIME INITIAL SET UP - DPPDRIFS
   ENDIF
   WHILE TOD CLOCK VALID (NOT IN ERROR HDWR)
01   IF TOD LT NEXT AND
01   IF TOD GT PREV
02     IF ERROR LT 1 SEC
03       IF TOD CLOCK TOO SLOW
04         CORRECT FOR DRIFT
03       ELSE TOD CLOCK MAY BE TOO FAST
04         SAVE ERROR AMT
04         IF 5 VALUES SAVED
05           GET MIN CORRECTION FACTOR FROM LAST 5
05           CORRECT FOR DRIFT
04         ENDIF
03       ENDIF
02     ENDIF
02     ADJUST TIMES FOR NEXT INTERRUPT
01   ELSE TOD CLOCK OUT OF RANGE
02     IF TOD CLOCK FAR IN FUTURE
03       ADVANCE EXPECTED TIMES
02     ENDIF
01   ENDIF
   BGNWHILE
01   WAIT FOR EXT INTERRUPT AND READ TOD CLOCK
   ENDDO
   ISSUE MESSAGE THAT TOD CLOCK INVALID (ERROR STATE OR NOT OPERATIONAL)
   WAIT FOREVER
   ENDSEGMENT DPPDRIFT
```

Figure 3-58. DPPDSUB2

```
    DPPDSUB2-DATA,BASE INTERFACE ROUTINE FOR 2 PARTITIONS
01    DPDSUB2A-ENTRY POINT FOR GETARRAY/PUTARRAY REQUESTS
02      PREPARE REGS TO EXECUTE MASTER PARTITION ROUTINES
02      BRANCH TO COMMON ROUTINE- DPDSUB2C
01    DPDSUB2I-ENTRY POINT FOR GETITEM/PUTITEM REQUESTS
02      PREPARE REGS TO EXECUTE MASTER PARTITION ROUTINES
02      BRANCH TO COMMON ROUTINE
01    DPDSUB2B-ENTRY POINT FOR GETBLOCK/PUTBLOCK REQUESTS
02      PREPARE REGS TO EXECUTE MASTER PARTITION ROUTINES
02      BRANCH TO COMMON ROUTINE
01    DPDSUB2G-ENTRY POINT FOR GETLOG REQUESTS
02      PREPARE REGS TO EXECUTE MASTER PARTITION ROUTINES
02      BRANCH TO COMMON ROUTINE
01    DPDSUB2P-ENTRY POINT FOR PUTLOG REQUESTS
02      PREPARE REGS TO EXECUTE MASTER PARTITION ROUTINES
02      BRANCH TO COMMON ROUTINE
01    DPDSUB2D-ENTRY POINT FOR DUMPLOG REQUESTS
02      PREPARE REGS TO EXECUTE MASTER PARTITION ROUTINES
02      BRANCH TO COMMON ROUTINE
01    DPDSUB2C-COMMON ROUTINE
02      IF   REQUEST IS VALID FOR 2 PARTITIONS
03        IF BRANCH ENTRY IS OK
04          SETPSW  TO GET PROTECT KEY 0
04          BALR  TO MASTER ROUTINES
03        ELSE
04          PATCH DATA BASE TASK IN MASTER PARTITION
04          WAIT  FOR COMPLETION
03        ENDIF
02      ELSE
03        SET RETURN CODE
02      ENDIF
01    RETURN
    ENDSEGMENT
```

Figure 3-59. DPPDUMPL

```
    DPPDUMPL-MAIN SEGMENT                  DUMP LOG ROUTINE
01    READ JFCB FOR SPECIFIED DUMPLOG DD NAME
01    IF     DISP=OLD CR NEW,THEN
02      IF  DUMPLOG REQUEST DISP=NEW,OR
02      IF  DISP=NEW,THEN
03        SET TO OPEN IT NEW BUT TO RESET TO MOD AFTER OPEN
02      ELSE
03        SET TO OPEN IT MOD
02      ENDIF
01    ENDIF
01    OPEN  DUMPLOG DCB
01    IF REPOSITIONING OF DUMPLOG DEVICE REQUIRED
02      IF  DEVICE IS A DISK
03        POINT TO START OF CATA SET
02      ELSE IT'S A TAPE
03        FEOV
02      ENDIF
01    ENDIF
01    UNTIL ALL ARRAYS HAVE BEEN PROCESSED
01    IF NUMBERED ARRAY
02      ARRAY ID = NUMBER
01    ELSE
02      FIND SALT PAGE FOR THIS ARRAY NAME
02      FIND ARRAY NAME IN SALT PAGE
02      CALCULATE ARRAY ID (ID=DISPLACEMENT INTO SALT TABLE/ SALT SIZE)
01    ENDIF
01    FIND  PALT
01    FIND  CURRENT LOG HEADER
01    BUILD DUMPLOG CUTPUT BUFFERS
01    READ  IN REQUESTEC LOG COPY
01    WRITE OUT LOG COPY TO DUMPLOG DATA SET
    ENDSEGMENT
```

FIGURE 3-60. DPPDUPDL

```
    DPPDUPDL - DATA BASE REFRESH
01  UNTIL ALL ELIGIBLE VS RESIDENT ARRAYS HAVE BEEN REFRESHED
02     LOCATE LAST CHECKPOINTED BLOCK NUMBER FOR THIS ARRAY
02     UNTIL MOST RECENT LOG COPY FOUND
03        GETBLOCK READ IN FIRST BLOCK OF THIS LOG COPY
02     ENDDO
02     GETBLOCK READ IN ENTIRE LOG COPY
01  ENDDO
    ENDSEGMENT DPPDUPDL
```

Figure 3-61. DPPDWRST

```
     MODULE NAME = DPPDWRST                                                  *
     DESCRIPTIVE NAME = DATA BASE WRITE/RESTART  OPEN/CLOSE ROUTINE'         *
     SAVE DPPIIRB ECB ADDRESS
     SAVE OPEN CLOSE FLAG
     RETRIEVE ADDRESS OF DA DDNAME TABLE
     STORE DBINIT DCB ADDR IN OPEN/CLOSE LIST
     BAL TO OPEN/CLOSE SEGMENT
     UNTIL ALL DA ARRAY DCBS IN DDNAME TABLE
01     ******* ARE  PROCESSED
01     LOCK THE DA ARRAY DCB UNTIL OPEN/CLOSE FINISH
01     STORE DCB ADDRESS IN OPEN/CLOSE LIST
01     BAL TO OPEN/CLOSE SEGMENT
01     UNLOCK THE DA ARRAY DCB
     ENDDO
     IF DBINIT DCB OPEN
01     BUILD DIRECTORY ENTRY FOR @INIT ARRAY
01     IF PRE-RESTART
02       *******DATA BASE AND POST-RESTART DATA BASE ARE DIFFERENT
02       ISSUE ERROR MESSAGE (SYSTEM MESSAGE 9)
01     ENDIF
     ENDIF
     BGNSEG *:OPEN/CLOSE SEGMENT
01     IF PRE-RESTART THEN
02       CLOSE DCB
01     ENDIF
01     IF POST-RESTART THEN
02       OPEN DCB
02       IF DCB NOT OPENED
03         ISSUE ABEND 12
02       ENDIF
01     ENDIF
     ENDSEG OPENCLSE
     ENDSEGMENT DPPDWRST
```

Figure 3-62. DPPFAONC

```
   ENTRY POINT IADDR
   ADCRESS RETURN CODE
   LOAC ADCRESS TO BE RETURNEC
   CLFAR UPPER BYTE
   SET RETURN CCDE
   INDICATE RETURN
   ENTRY POINT CRBIT
   LCAD LCCATICN TO ALTER
   LOAC ADCRESS OF FLAGS TO CR
   SET REQUESTED FLAGS
   SET RETURN CCDE
   INCICATE RETURN
   ENTRY POINT NCBIT
   AND BIT CCDE
   LOAD LOCATICN TO ALTER
   LOAC ADCRESS CF FLAGS TO ANC
   DELETE DESIRED FLAGS
   SET RETURN CODE
   INDICATE RETURN
   ENTRY PCINT COPY
   DATA MOVER CCDE
   HOUSEKEEPING
   GET PCINTER TO INPUT ADDRESSES
   UNTIL PARMS EXHAUSTED DO
01   GET ADDRESS OF INFUT AREA THUS
01   IF INPUT ADDRESS IS NCN-ZERO AND SO
01   GET ADDRESS OF CUTPUT AREA THUS,
01   IF OUTPUT ADCRESS IS NCN-ZERO THEN
02     IF INPUT GT OUTPUT OR
02     IF INPUT IS NOT GT ZERC TFEN
03       USE OUTPUT SIZE
02     ENDIF
02     CCPY SIZE CF MCVE
02     MOVE DATA TO OUTPUT FIELD
01   ENDIF
01   INCREMENT TC NEXT PCINTER
01   INCREMENT TO NEXT POINTER
   ENDDO
   RESTORE CALLERS REGISTERS
   RETURN EQUALS ZERO
   INDICATE RETURN
   RETURN TC CALLER
```

Figure 3-63. DPPFIXFR

```
     DPPFIXFR MAIN SEGMENT
     IF ADDRESSES WITHIN PARTITICN
01     IF DPPFREE
02       ISSUE PGFREE
01     ELSE CPPFIX
02       UNTIL PAGES FIXED OR NOT POSSIBLE
03         ISSUE PAGE FIX
03         IF WAIT NEEDED
03         IF
04           WAIT CN PAGING TASK
04           IF PURGE TOCK PLACE
05             SET TO TRY AGAIN
04           ELSE
05             IF ERRCR CODE
06               SET RETURN ERROR
05             ELSE
06               SET NC ERROR RETURN
05             ENDIF
04           ENDIF
03         ELSE
04           IF SUCCESFULL
05             SET GCOD RETURN CCDE
04           ELSE
05             SET ERROR RET CCDE
04           ENDIF
03         ENDIF
02       ENDDO
01     ENDIF
     ELSE
01     ABEND CALLER
     ENDIF
     ENDSEGMENT CPPFIXFR
```

Figure 3-64. DPPIDBAS

```
   DPPIDBAS—MAIN SEGMENT
    INITIALIZE WORK AREA
    GET PROTECT KEY ZERO
01    IF     SLAVE PARTITION INITIALIZATION
02      COPY      CPIDBAS6
01    ELSE
    COPY DPIDBAS1                     READ INITIALIZATION DATA
    IF NC ERRORS OCCURED
01    COPY DPICBAS2                   BUILD TABLE HEADERS
01    IF NC ERRORS OCCURED
02      CCPY DPICBAS3                 CHAIN TABLES TOGETHER
01    ENDIF
    ENDIF
01    ENDIF
    RESET PRCTECTICN KEY
    IF ERRCRS HAVE OCCURED
01    ABEND STEP WITH USER CCMPLETICN CODE
    ELSE
01    LOAD  GETARRAY,GETITEM, AND GETBLOCK ROUTINES
01    STORE ROUTINE ADCRESSES INTO THE SCVT
01    STORE PAGE BCUNCARY TABLE ADCRESS INTO THE SCVT
01    STORE ARRAY LOCATOR TABLE ADDRESS INTO TFE SCVT
   *        NOTE  THIS STORE IS DCNE WITH A PCST SVC IN CASE OF
   *              SYNCRONIZATION OF 2 PARTITION OPERATION
    ENDIF
   ENDSEGMENT DPPICBAS
```

Figure 3-65. DPPIIRB

```
MODULE NAME = DPPIIRB                                                 *
DESCRIPTIVE NAME = CATA BASE CREATE IRB RCUTINE                       *
PLACE DPPIIRB PSW IN SUPERVISOR STATE
SAVE DPPIIRB CRIGINAL PSW STATE FLAGS
CREATE AN IRB FOR DPPDWRST
SCHEDULE DPPDWRST IRB UNDER JOB STEP TCB
WAIT UNTIL DPPDWRST HAS OPENED OR CLOSED
THE CATA BASE DCBS
RESTORE DPPIIRB PSW TO ORIGINAL STATE
IRB ENTRY SEGMENT
SAVE IRB ECB ADCRESS
LINK TO OPEN CLOSE ROUTINE
POST DPPIIRB ECB
ENDSEGMENT  CPPDIRB
```

Figure 3-66. DPPILOGN

```
    DPPILOGN - DATA BASE LOGGING INITIALIZATION
01  IF MASTER PARTITION BEING INITIALIZED
02     IF PRE-RESTART THEN
03        LOAD DPPDGETC GETLOG ROUTINE
03        LOAD DPPDPUTL PUTLOG ROUTINE
03        LOAD DPPDUMPL DUMPLOG ROUTINE
03        INITIALIZE LOGGING CONTROL BLOCKS (LCB)
03        BUILD ARRAY ID LIST OF EACH LOG FREQUENCY
03        IF REFRESH REQUESTED THEN
04            LINK TO DPPDUPDL-REFRESH VS ARRAYS
03        ENDIF
03        ISSUE PTIMES ON SPECIFIED LOG FREQUENCY TO DPPDFREQ
02     ELSE
03        IF REFRESH REQUESTED THEN
04            LINK TO DPPDUPDL-REFRESH VS ARRAYS
03        ENDIF
02      ENDIF
01    ENDIF
    ENDSEGMENT DPPILOGN
```

Figure 3-67 (1 of 2). DPPINIT

```
   ***************************--  DPPINIT  --******************************
   *                                                                      *
   *                          **************************                  *
   *                          *                        *                  *
   *                          *     INITIALIZATION     *                  *
   *                          *                        *                  *
   *                          **************************                  *
   *                                                                      *
   *        DPPINIT - INITIALIZATION ROUTINE                              *
   *                                                                      *
   ************************************************************************
   IF DPPINIT NOT JOB STEP TASK
01   ABEND THE JOB STEP - USER 30
   ENDIF
   CALL DPPINITO - CARD READ ROUTINE
   IF TCB CARD READ
01   UPDATE #TCBS FOR THIS RUN
   ENDIF
   IF CBGET CARD READ
01   UPDATE AMOUNT OF CBGET CORE REQUIRED
   ENDIF
   IF GETWA CARD READ
01   OVER-RIDE GETWA TABLE
   ENDIF
   GET  PAGEABLE KEY ZERO CORE (SP-253) FOR XCVT AND SCVT                 *
   CLEAR XCVT CORE
   INITIALIZE XCVT
   CLEAR SCVT CORE
   INITIALIZE SCVT                                                        *
   IF DBREF = YES WAS REQUESTED
01   TURN ON REINIT FLAG
   ENDIF
   GET TCB ADDR AND GO TO RBX TO GET PARTN HI AND LO ADDRESSES
   PUT JOB STEP TCB ADDR IN SCVT
   IF NOT TWO PARTITION OPERATION
01   LEAVE TWO PARTITION FLAG OFF
   ELSE
01   TURN ON TWO PARTN OPERATION FLAG
01   IF THIS IS A MASTER PARTITION
02     TURN ON SCVT MASTER PARTN FLG
02     TURN ON XCVT MASTER PARTN FLG
01   ELSE
02     TURN ON SCVT SLAVE PARTN FLG
02     TURN ON XCVT SLAVE PARTN FLG
01   ENDIF
```

Figure 3-67 (2 of 2). DPPINIT

```
   ENDIF
   IF PROBE OR -
   IF WRITE RESTART
01   TURN ON READ-ONLY TEMPORARY FLAG
   ENDIF
   IF EXTERNAL INTERRUPT INITIALIZATION OR RESTART PROCESSING REQUIRED
   LINK TO DOMIRINT TO PERFORM EXTERNAL INTERRUPT INITIALIZATION
   *       AND/OR PRE-RESTART PROCESSING.
   COPY DPINIT1 - TMCT INITIALIZATION
   COPY DPINIT2 - CBGET INITIALIZATION
   COPY DPINIT3 - TASK MGMT INITIALIZATION
   LOAD DPPIPFRE - PAGE UNFIX ROUTINE
   ISSUE STAE FOR JOB STEP TASK
   ATTACH PROGRAM DPPINIT1
   INIT COMPLETE - XCTL TO DPPTSMON
   **    TWO PARTN TIMER EXIT ROUTINE
   **    POST TIMER ECB TO ISSUE MSG
   **       FOR TWO PARTN SYNCHRONIZE
   RETURN
   COPY DPINIT4 - DATA AREAS
```

Figure 3-68 (1 of 2). DPPINITO

```
    DPPINITO - CCNTROL STATEMENT PROCESSOR          INITIALIZATION         *
    TUPN CN DBREINIT FLAG
    IF TWO PARTITICN CPERATICN ALLOWED
01    TURN CN TWC PARTN ALLOWED FLAG
    ENDIF
    OPEN INPUT CCB
    IF DCB FAILED TO OPEN
01    WRITE ERR MSG
01    SYSINIT DCB FAILED TO OPEN ABEND 40
    ENDIF
    UNTIL ENDLESS LCCP - EXIT BY EOCAD
01    CPEN OUTPUT CCB
01    IF CCB CPEN
02      WRITE CARD IMAGE
01    ENDIF
01    IF MCRE OPERANCS ARE EXPECTED
02      IF A CCNTINUATICN WAS NOT EXPECTED
03        IF NCN BLANK CHAR FCUND
04          IF NCN BLANK IN POS 1 - NAME FLD
05            IF THIS IS NOT CCMMENT CARD
06              IF NAME LE 8 CHARS
07                MCVE NAME TC WA
06              ELSE
07                WRITE ERROR MSG
07                GC BACK AND READ ANOTHER CARD
06              ENDIF
05            ELSE
06              GC READ ANOTHER CARD
05            ENDIF
05            LCCK FCR CPERATION FIELD
04          ENDIF
04          IF CPN LE 6 CHAR
05            MCVE CPERATICN TC WORK AREA
05            SEARCH FOR OPERANDS
05            IF OPND ON THIS CARD
06              IF CPERANDS END BEFORE LAST CATA COL
07                IF DELIMITER A BLANK
08                  IF MCRE CPNCS EXPECTEC
09                    TURN CN CCNT EXP FLG
08                  ELSE
09                    IF CCNTINUATION EXPECTED
10                      TURN CN THE CCNTINUATION AND NO MORE OPERANDS FLAG
09                    ELSE
10                      TURN OFF THE CONTINUATION AND NO MORE OPERANDS FLA
09                    ENDIF
08                  ENCIF
07                ELSE
08                  BREAK OUT BLANKS (DC DPINITC6)
07                ENDIF
```

Figure 3-68 (2 of 2). DPPINITO

```
06             ELSE
07               IF NO CONT EXP
08                 TURN OFF CCNTINUATION FLG
07               ELSE
08                 TURN CONTINUATION FLAG ON
07               ENDIF
06             ENDIF
06             MOVE TFE OPERANDS TO THE WA
06             UPCATE OPND WA FOR NEXT MOVE
06             SAVE COUNT OF OPNDS MOVED
05           ELSE
06             IF NCT ABEND CARD
07               WRITE ERROR MESSAGE
06             ENDIF
05           ENDIF
04         ELSE
05           WRITE ERRCR MESSAGE
04         ENDIF
03       ELSE
04         WRITE ERROR MESSAGE
03       ENDIF
02     ELSE
03       COPY DPINIT03 - CONTINUATION CARD PROCESSOR
02     ENDIF
01   ELSE
02     IF NO CCNTINUATION
03       TURN OFF CONTINUATICN AND NO OPERANDS FLAGS OFF
02     ENDIF
01   ENDIF
01   COPY DPINIT04 - CCNTROL CARD PROCESSOR
   ENDDO
   COPY DPINITC5 - BUILD WAIT LIST RTNE
   COPY DPINIT08 - INITCB BUILC AND CHAIN RTNE
   COPY DPINIT09 - ERRCR MSG WRITER RTNE
   COPY DPINIT06 - BLANKS IN PARAM PROCESSOR
   COPY DPINITC7 - DATA AREAS AND CONSTANTS
```

Figure 3-69 (1 of 3). DPPINIT1

```
   DPPINIT1 - SUBSYSTEM PATCHCR RCUTINE                                              *
   IF 2  PTN OPERATICN
01   IF THIS IS THE SLAVE PTN
01   ENDIF
   ENDIF
   IF INITCB'S EXIST
01   UNTIL END CF CHAIN
02     IF THIS IS A PATCH BLOCK
03       GET PROBL ADDR
03       CET TFE PRCBL LENGTH
03       IF PARAM CCDED ON PATCH STATEMENT
04         PATCH THE PROGRAM WITH ECB SPECIFIED
03       ELSE
04         IF THIS IS A PRE-WRITE RESTART PATCH
05           IF A WRITE RESTART BLCCK EXISTS
06             PATCH TFE PROGRAM WITH ECB SPECIFIED
05           ELSE
06             PATCH PROGRAM WITH NO ECB
05           ENDIF
04         ELSE
05           PATCH PRCGRAM WITF NO ECB
04         ENDIF
03       ENDIF
03       IF PATCF FAILED
04         ABEND WITH USER 31
03       ENDIF
02     ELSE
03       IF THIS IS A WAIT BLOCK
04         IF WAIT IS TO BE ON A LIST
05           GET WAIT COUNT
05           GET ADDR OF ECB LIST
05           WAIT CN LIST
05           UNTIL CFECK ECB PCST CODES
06             IF POST CODE IS NCN ZERC
07               DO DPINIT11 - WRITE ERROR MSG
06             ENDIF
05           ENDDO
04         ELSE
05           GET ECB ADDR
05           WAIT ON ECB
05           IF POST CODE IS NCN ZERO
06             DC DPINIT11 - WRITE ERROR MSG
05           ENDIF
04         ENDIF
```

Figure 3-69 (2 of 3). DPPINIT1

```
03        ELSE
04          IF THIS IS AN ABEND BLOCK
05            WAIT FOR INTERVAL IN ECB
05            IF A DUMP IS REQUESTED
06              ABEND WITH A DUMP - USER 22
05            ELSE
06              ABEND WITHOUT A DUMP - USER 22
C5            ENDIF
04          ELSE
05            THIS MUST BE A RESTART CARD
05            GET WAIT COUNT
05            IF ANY PATCH CARCS PRECEED WRITE RSTRT
06              GET ACDR OF WAITLIST
06              WAIT ON LIST
06              UNTIL CHECK ECB POST CODES
07                GET ECB ADDR
07                TURN OFF POST BIT
07                IF POST CODE IS NCN ZERO
08                  DO DPINIT11 - WRITE ERROR MSG
07                ENDIF
06              ENDDO
06              IF ANY POST CODES WERE NON ZERO
07                ABEND WITH A USER 35
06              ENDIF
05            ENDIF
05            IF THIS IS A SLAVE PARTITION
06              IF WRITE RESTART REQUEST
C7                IF SLAVE IS BEING RESTARTEC - (RE-SYNC)
08                  ISSUE WTFAILDS BYPASSED MSG
07                ELSE
08                  ISSUE WTFAILDS
07                ENDIF
06              ENDIF
06              IF CANCEL RECUEST AND
06              IF INITIAL IPL
06              ENDIF
05            ELSE
06              IF WRITE RESTART REQUEST
07                WRITE FAILOVER RESTART CATA SET
07                IF DR #5626
07                ENDIF DR#5626
06              ENDIF
06              IF CANCEL REQUEST AND
06              IF INITIAL IPL
06              ENDIF
06              IF PROBE REQUEST
07                IF DR #5626
07                ENDIF DR#5626
06              ENDIF
06              IF
```

Figure 3-69 (3 of 3). DPPINIT1

```
06              ENDIF
05            ENDIF
05            IF PROBE OR
05            IF WRITE RESTART
05            ENCIF
05            UNTIL END OF CHAIN
06              IF THIS IS A PATCH BLOCK
07                GET PROBL ACDR
07                ADJUST RESTART FLAGS
06              ENDIF
05            ENDDO
04          ENDIF
03        ENDIF
02      ENDIF
01    ENDDO
01    UNTIL LOOP FREEING  ALL GCTTEN CORE
02      IF THIS IS A PATCH BLOCK
02      ELSE
03        IF THIS IS A WRITE RESTART BLOCK
04          FREE WAIT LIST
03        ELSE
04          THIS IS A WAIT BLOCK
04          IF A WAIT BLOCK EXISTS
05            GET THE LENGTH OF THE LIST
05            FREEMAIN THE LIST
04          ENDIF
03        ENDIF
02      ENDIF
02      FREE THE CCNTROL BLOCK (INITCB)
01    ENDDO
    ENCIF
    TURN CN END OF INITIALIZATICN FLAG
    RETURN TO CALLER
    COPY DPINIT11 - ERRCR ROUTINE
    COPY DPINIT12 - PRCBE ROUTINE
    COPY CPINIT13 - CMCN ROUTINE
```

Figure 3-70 (1 of 2). DPPIPFIX

```
   DPPIPFIX  -   INITIALIZATION PAGE FIX ROUTINE                        *
   IF ARRAY DPPXFIX EXISTS
01   WHILE PROCESS ARRAY
01   WHILE AND FIX STORAGE TILL END OF ARRAY
02     IF FIX REQUEST IS FOR A LOAD MODULE
03       IF STORAGE NOT ALREADY FIXED
04         FIND THE LOAD MODULE - BLDL
04         IF MODULE FOUND - LOAD IT
05           IF A FIX LENGTH WAS REQUESTED
06             ADD REQUESTED LENGTH TO EP
05           ELSE
05           ENDIF
05           FIX THE VIRTUAL STORAGE
05           IF FIX SUCCESSFUL
06             SAVE LOW FIX ADDR IN THE ARRAY
06             SAVE HI  FIX ADDR IN THE ARRAY
05           ELSE
06             ISSUE ERROR MSG
05           ENDIF
04         ELSE
05           ERROR MSG
04         ENDIF
03       ENDIF
02     ELSE
03       IF THIS IS A NAMED ARRAY FIX REQUEST
04         IF STORAGE NOT ALREADY FIXED
05           FIND THE NAMED ARRAY
05           IF ARRAY FOUND
06             GET THE ARRAY ADDR
06             IF A LENGTH WAS REQUESTED
07               GET HI FIX ADDR
06             ELSE
07               IF BLK CNT NONZERO
08                 GET THE BLOCK SIZE
08                 MULT BLK CT X BLK SIZE = ARRAY SIZE
07               ELSE
08                 GET ARRAY SIZE
07               ENDIF
06             ENDIF
06             FIX THE VIRTUAL STORAGE
06             IF FIX SUCCESSFUL
07               PUT THE LO FIX ADDR IN THE ARRAY
07               PUT THE HI FIX ADDR IN THE ARRAY
06             ELSE
07               ISSUE ERROR MESSAGE
06             ENDIF
05           ELSE
06             ISSUE ERROR MESSAGE
05           ENDIF
04         ENDIF
```

Figure 3-70 (2 of 2). DPPIPFIX

```
03       ELSE
04         IF THIS IS A NUMBERED ARRAY FIX REQUEST
05           IF STORAGE NOT ALREADY FIXED
06             FIND THE ARRAY
06             IF ARRAY FOUND
07               GET THE ARRAY ADDR
07               IF A LENGTH WAS REQUESTED
08                 GET HI FIX ADDR
07               ELSE
08                 IF BLK CNT NONZERO
09                   BLK CVT X BLK SIZE = ARRAY SIZE
09                   GET ARRAY SIZE
08                 ELSE
08                 ENDIF
07               ENDIF
07               FIX THE VIRTUAL STORAGE
07               IF FIX SUCCESSFUL
08                 PUT LOW FIX ADDR IN ARRAY
08                 PUT HI  FIX ADDR IN ARRAY
07               ELSE
08                 ISSUE ERROR MESSAGE
07               ENDIF
06             ELSE
07               ISSUE ERROR MESSAGE
06             ENDIF
05           ENDIF
04         ELSE
05           ISSUE PAGE FIX STATUS MSG
04         ENDIF
03       ENDIF
02     ENDIF
01   ENDDO
01   IF ALL FIXES OK
01   ELSE
01   ENDIF
01   ISSUE ERROR MSG
   ELSE
   ENDIF
   RETURN
```

Figure 3-71. DPPIPFRE

```
   DPPIPFRE   -   PACE UNFIX ROUTINE
   GETARRAY - FINC ARRAY DPPXFIX
   IF ARRAY FOUND SUCCESSFULLY
01   WHILE *    PROCESS ALL ITEMS
01   WHILE *       IN ARRAY - FIXING VIRTUAL
02     *           ADDRESSES
02     IF PREV FIX SUCCESSFUL
03       UNFIX ALL VIRTUAL ADDRESS
02     ENCIF
02     UPCATE TO NEXT ITEM IN ARRAY
01   ENDDO
   ENDIF
   RETURN
```

Figure 3-72. DPPISTAE

```
   DPPISTAE   -   STAE RCUTINE
   GET CVT ADDR
   GET PTR TO NEW/CLD WORDS
   GET CURRENT TCB ADDR
   GET TCBX ADCR
   GET XCVT ADDR
   IF
01   GET PAGE FREE (UNFIX) RTNE ACCR
01   LINK TO PAGE UNFIX
   ENDIF
   IF THE EXT INT HANDLER IS INITIALIZED
01   TURN CFF EXT INT HANDLER BITS
   ENDIF
   IF THIS IS A TWO PARTITICN RUN
01   IF CTHER JOB STEP TASK IS NOT ABENDING
02     IF THIS IS THE MASTER PARTITION
03       ABEND THE SLAVE - USER 41
02     ELSE
03       TURN OFF THE 2 PARTN FLG
02     ENDIF
01   ENDIF
   ENDIF
   RETURN
```

Figure 3-73. DPPITIMI

```
    DPPITIMI-MAIN SEGMENT                  'TIME MANAGEMENT INITIALIZATION'
01    INITIALIZE  DPPITIME WORK AREA
01    GET ARRAY   DPPCTIMA                 TIME DATA BASE ARRAY
01    SETPSW      TO GET CINFO ADDRESS
01    STORE       ARRAY ADDRESS IN SCVT
01    DEFLOCK     TIME                     BUILD LOCK CONTROL BLOCK
01    SAVE  JOB STEP'S PRIORITY
01    INITIALIZE FAILOVER FIELDS IN TIME ARRAY
01    STORE       TOD CLOCK                SET THE CURRENT TOD COUNTER VALUE
   *                                       INTO THE TIME ARRAY
01    LINK        DPPCALCF                 CALCULATE THE CORRECTION FACTOR
01    LINK        DPPCUPCF                 CALCULATE THE CONVERSION FACTOR
01    IF          TOD CLOCK OPERATIONAL
02      ATTACH    DPPCPTIM                 ISSUES PATCHES REQUESTED VIA PTIME
02      IF  CLOCK COMPARATOR IS AVAILABLE
03        ATTACH DPPCTIM2
02      ELSE
03        ATTACH  DPPCTIME
02      ENDIF
02      WAIT UNTIL THE TWO TASK ARE INITIALIZED
01    ELSE
02      ABEND JOB STEP                     ABORT INITIALIZATION
01    ENDIF
    RETURN
```

Figure 3-74. DPPMINIT

```
    DPPMINIT MAIN SEGMENT *SYSTEM MESSACES INITIALIZATION ROUTINE *
01    PATCH DPPMMSG1 WITH ID 255
01    GETARRAY FOR ROUTING CODE TABLE(ARRAY DOMXSMRC)
01    IF RCUTING CODE TABLE NCT FOUND THEN
02      ABEND TASK WITH ID 23
01    ENDIF
01    GETMAIN FOR MESSAGE CONTFOL TABLES
01    MOVE MESSAGE CCB TO MESSAGE DCB TABLE(MDT)
01    OPEN MESSAGE CCB FOR INPUT
01    IF MESSAGE CCB NCT OPENED THEN
02      ABEND TASK WITH ID 20
01    ENDIF
01    IF PRE-RESTART THEN
02      SET PRE-RESTART FLAG IN MESSAGE ADDRESS TABLE
01    ELSE
02      SET POST-RESTART FLAG IN MESSAGE ADDRESS TABLE
01    ENDIF
01    IF CCNAMES SPECIFIED IN ROUTING CCDE TABLE
02      DC UNTIL ALL CUTPUT DCB'S ARE OPENED
03        MOVE DDNAME FRCM ROUTING CODE TABLE TO DUMMY DCB
03        MOVE FCRMATTED CCB TO MESSAGE CCB TABLE
03        OPEN FORMATTED DCB FCR OUTPUT
03        IF FORMATTED DCB NOT OPENED THEN
04          ISSUE WTO ERRCR MESSAGE
03        ENDIF
02      ENDDO
01    ENDIF
01    LOAD REAL TIME MESSAGE HANDLER(DPPMMSG) IN CORE
01    STORE DPPMMSG ADCRESS IN SCVT(SCVTMSGH)
01    BUILD LCCK CONTRCL BLOCK FOR DPPMMSG
01    STORE ADDRESS CF LOCK CONTROL BLCCK IN MESSAGE ADDRESS TABLE
01    BUILD LCCK CCNTRCL BLOCK FOR MESSAGE OUTPUT RCUTINE(DPPMMSG1)
01    STORE ADCRESS OF LOCK CCNTROL BLCCK IN MESSAGE ADDRESS TABLE
01    STORE ADCRESS OF MESSAGE ADDRESS TABLE IN SCVT(SCVTMWA)
    ENDSEGMENT DFFMINIT
```

Figure 3-75. DPPMMSG

```
    DPPMMSC  MAIN SEGMENT * REAL TIME MESSAGE HANDLER FORMATTER ROUTINE *
01    GFT MESSAGE REQUEST FRCM USER
01    GET ADDRESS OF MESSAGE ADDRESS CCNTROL BLOCK FROM SCVT
01    GFT ADDRESS CF LCCK CONTROL BLOCK FROM MESSAGE ADDRESS CONTRCL BLOCK
01    LCCK DPPMINIT
01    GET ADDRESS OF OPEN MSG CCB FROM MSG ACDRESS CCNTROL BLOCK
01    GET SPECIFIEC MESSAGF FRCM MESSAGE CATA SET
01    IF MESSACE NCT FCUND THEN
02      SET RFTURN CCDE TO 8
01    ELSE
02      IF VARIABLE CCUNT IN MESSAGE PARAMETER < VARIABLE CGUNT IN DEFINED
03           MESSACE THEN
02        SET RETURN    CODE TC 4
01      ELSE
02        IF VARIABLE COUNT IN MESSACE PARAMETER > VARIABLE CCUNT IN
03              DEFINED MESSAGE THEN
02          SET RETURN CODE TO 2
01        ENDIF
        ENCIF
        CONVERT ALL USER DATA(VARIABLES) TO EBIDIC INTO MESSAGE
        ADC TIME TO MESSAGE
        IF DATE FLAG CNIN DEFINED MESSAGE
01        ADD CATE TC MESSAGE
        ENDIF
        IF ACTICN CODE SPECIFIED IN PARAMETER LIST THEN
01        MOVF ACTICN CODE TO MESSAGE
        ENDIF
        IF USER MESSAGE RETURN AREA SPECIFIED THEN
01        PASS MESSAGE TO USER SPECIFIEC AREA
        ENDIF
        IF ROUTE CODE NOT EQUAL TO 255 THEN
01        RETRIEVE ROUTING CODE TABLE FRCM CATA BASE
01        IF ROUTE CODES IN PARAMETER LIST NCT FOUND IN ROUTING CODE TABLE
02             THEN
01          SET RETURN CCDE TC 12
          ELSE
01          PATCH MESSAGE OUTPUT ROUTINE
          ENDIF
        ENDIF
      ENDIF
      UNLCCK DPPMMSG
     ENDSEGMENT DPPMMSG
```

Figure 3-76. DPPMMSGV

```
      DPPMMSGV MAIN SEGMENT * SYSTEM MESSAGE ROUTING CODE MANIPULATION
01                                ROUTINE *
        RETRIEVE PASSED PARAMETERS
        RETRIEVE MESSAGE ROUTING CODE FROM DATA BASE
        IF ALTERNATE ROUTE CODE PASSED THEN
01        DO UNTIL ALTERNATE ROUTE CODE COMPARED AGAINST ROUTE CODE TABLE
02          IF ALTERNATE ROUTE CODE OUT OF SERVICE THEN
03            ISSUE ERROR MESSAGE
03            SET ERROR FLAG
02          ENDIF
01        ENDDO
        ENDIF
        IF PASSED PRIMARY ROUTE CODE = ALTERNATE ROUTE CODE THEN
01        SET ERROR FLAG
01        ISSUE ERROR MESSAGE
        ENDIF
        IF ERROR FLAG OFF THEN
01        DO UNTIL END OF ROUTE CODE TABLE
02          IF PRIMARY ROUTE CODE FOUND IN ROUTE CODE TABLE THEN
03            IF ALTERNATE ROUTE SPECIFIED THEN
04              MOVE ALTERNATE ROUTE CODE TO ROUTE CODE TABLE
03            ENDIF
03            IF ROUTE CODE OUT OF SERVICE THEN
04              SET OUT-OF-SERVICE  FLAG IN ROUTE CODE TABLE
04              DISPLAY STATUS OF ROUTE CODE
03            ENDIF
03            IF ROUTE CODE IN SERVICE THEN
04              SET IN-SERVICE FLAG IN ROUTE CODE TABLE
04              DISPLAY STATUS OF ROUTE CODE
03            ENDIF
03            IF STATUS OF ROUTE CODE REQUESTED THEN
04              DISPLAY STATUS OF ROUTE CODE
03            ENDIF
02          ELSE
03            IF STATUS OF ALL ROUTE CODES  IN SYSTEM REQUESTED THEN
04              DISPLAY STATUS OF ROUTE CODE(S)
03            ENDIF
02          ENDIF
01        ENDDO
        ENDIF
        IF BAD PARAMETER PASSED THEN
01        ISSUE ERROR MESSAGE
        ENDIF
      ENDSEGMENT DPPMMSGV
```

Figure 3-77. DPPMMSG1

```
    DPPMMSG1 MAIN SEGMENT *REAL TIME MESSAGE HANDLER OUTPUT ROUTINE *
01    GET FORMATTED MESSAGE AND ROUTE CODES FRCM MESSAGE FORMATTER ROUTINE
01    GET ADDRESS OF MESSAGE ADDRESS CONTROL BLOCK FROM SCVT
01    LOCK DPPMMSG1
01    IF PRE-RESTART FLAG SET IN MESSAGE ADDRESS CCNTROL BLOCK THEN
02      OUTPUT MESSAGE TO SYSTEM CCNSOLE
01    ELSE     IF PCST-RESTART
02      RETRIEVE MESSAGE ROUTING CODE TABLE FROM DATA BASE
02      DC UNTIL  MESSAGE HAS BEEN ROUTED TO ALL SPECIFIED ROUTE CODES
03        IF ROUTE CCDE FCUND IN ROUTING CODE TABLE THEN
04          OUTPUT MESSAGE TO SPECIFIED DEVICE IN ROUTING CODE TABLE
04          IF MESSAGE NOT OUTPUTTED TO PRIMARY ROUTE THEN
05            OLTPUT MESSAGE TO ALTERNATE ROUTE CODE
04          ENDIF
03        ENDIF
02      ENDDO
01    ENDIF
01    UNLOCK DPPMMSG1
    ENDSEGMENT CPPMMSG1
```

Figure 3-78 (1 of 2). DPPPARM

```
   A(PARAMETER TABLE)
   CANCFL THE PL/1 SPIE AND STAE EXIT ACROSS INTERFACE ROUTINE
   LOAD A(STAE REMOTE LIST)
   SAVE A(PL/1 PICA)
   A(CVT)
   A(NEW/CLD TCB ACCRESSES)
   A(CURRENT TCB)
   A(TCBX)
   A(PATCH PARM TABLE)
   IS THIS THE FIRST ENTRY
   IF *: NO
01   LOAD A(XCVT)
01   LOAC A(SCVT)
01   LOAD A(SEC. EP DPPTPMON)
01   LOAD POST CCDE FOR PMON TO PATCH
01   GO GET ANCTHER WORK QUEUE
   ELSE
01   CONTINUE RETURN CODE
   ENDIF
   FILL THE PARAMETER TABLE
   STORE RETURN CODE
   RESTORE PL/1 SPIE AND STAE PRIOR TO RETURNING
   RESTORE PL/1 STAE
   LOAD A(PL/1 PICA)
   RESTORE PL/1 SPIE EXIT
   STAE SUBRCUTINE TC PERMIT RECOVERY TO FREE CORE AND ENTER
   PL/1 STAE
   BGNSEG
01   IF THERE IS A WCRKAREA
02     LCAC A(DPPPARM SAVEAREA)
02     SAVE ABEND CCDE FCR RECOVERY
01   ELSE
02     SAVE ABENC CODE FCR RECOVERY
01   ENDIF
01   A(DPPPARM STAE RETRY ROUTINE)
01   REQUEST A RETRY
   ENDSEG CPARMSTA
   BGNSEG
01   IF THERE IS A WCRKAREA
02     LCAD A(DPPPARM SAVEAREA)
02     LOAD COMPLETICN CODE
02     LOAC STAE WORKAREA LENGTH
02     FREE THE STAE WORKAREA
02     SETUP A(DPPPARM SAVEAREA)
02     LCAD A(PL/1 SAVEAREA)
```

Figure 3-78 (2 of 2). DPPPARM

```
01    ELSE
02      SAVE TFE CCMPLETICN CODE
02      LOAC ADCRESS OF DPPPARM
02      LOAC A(CVT)
02      LOAC A(TCB NEW/OLD POINTERS)
02      LCAD A(CURRENT TCB)
02      LCAC A(FIRST SAVEAREA)
02      STRTSRCH NEXT IS NCT ZERO
02      EXITIF NEXT IS CPPPARM SAVEAREA
03        LCAC A(DPPPARM SAVEAREA)
02      ENCLOOP
02      ENDSRCH
01    ENDIF
01    LOAC A(PL/1 PICA)
01    LOAD SAVEAREA LENGTH
01    FREE TFE SAVE AREA
01    SETUP TO RESTORE PL/1 SPIE
01    RESTCRE PL/1 SPIE EXIT
01    COMPLETICN CODE TO REG. 1
01    SETUP PL/1 SAVEAREA REG.
01    RESTORE PL/1 REGS 2 TO 12
01    GO TO PL/1 STAE EXIT
    ENDSEG DPARMRCV
```

Figure 3-79 (1 of 5). DPPPIF

```
   A(FIRST PARAMETER LIST)
   OVERRIDE PL/I STAE EXIT DURING CALLS FOR SERVICE
   LOAD A(STAE REMOTE LIST)
   CANCEL PL/I SPIE EXIT DURING CALLS FOR SERVICE
   SAVE A(PL/I PICA)
   A(CVT)
   A(NEW/OLD TCB ADDRESSES)
   A(CURRENT TCB)
   A(JOBSTEP TCB)
   A(TCBX)
   A(XCVT)
   MOVE ERROR MSG SKELETON TO CS
   IF MACRO NO. IS MULTIPLE OF 4 AND
   IF ITS NOT NEGATIVE AND
   IF ITS NOT GT LIST, THEN
01   EXEC. INTERFACE CODE
   ENDIF
   STORE RETURN CODE/POST FLAGS
   RESTORE PL/I STAE
   LOAD A(PL/I PICA)
   RESTORE PL/I SPIE EXIT
   THIS SUBROUTINE SAVES THE COMPLETION CODE IN THE INTERFACE
   ROUTINE SAVEAREA AND REQUESTS THE RETRY OPTION.
   BGNSEG
01   IF WORKAREA THEN
02     LOAD A(DPPPIF SAVEAREA)
02     MOVE COMPLETION TO FIRST WORD
01   ELSE
02     STORE COMPLETION CODE
01   ENDIF
01   A(STAE RETRY ROUTINE)
01   REQUEST A RETRY
   ENDSEG DPIFSTAE
   THIS ROUTINE FREES THE STAE WORKAREA, THE SAVEAREA OBTAINED BY
   THE BEGIN MACRO AND FORCES A PL/I STAE ENTRY.
   BGNSEG
01   MAKE ENTRY THE BASE
01   IF REGISTER 1 CONTAINS A(WORKAREA)
02     SAVE STAE WORKAREA ADDRESS
02     LOAD A(INTERFACE SAVEAREA)
02     POINT TO ADRESS OF ABEND
01   ELSE NO WORKAREA - FIND SAVEAREA FROM TCB
02     A(CVT)
02     A(NEW/OLD TCB ADDRESSES)
02     A(CURRENT TCB)
02     A(FIRST SAVEAREA)
02     STRTSRCH TO PREV SAVEAREA
02     EXITIF PREV. SAVEAREA FOUND
03       LOAD A(INTERFACE SAVEAREA)
03       NO ABEND ADDRESS
02     ENDLOOP
02     ENDSRCH
01   ENDIF
```

Figure 3-79 (2 of 5). DPPPIF

```
01     POINT TO COMP. CODE FOR MSG
01     LOAD LANGUAGE SAVEAREA ADDRESS
01     LOAD ADDR OF PARAMETER ADDR
01     LOAD ADDR OF SERVICE ID FOR MSG
01     LOAD CVT ADDR
01     LOAD NEW/OLD TCB ADDR
01     LOAD CURRENT TCB ADDR
01     LOAD JOBSTEP TCB ADDR
01     LOAD TCBX ADDR
01     LOAD XCVT ADDR
01     IF A STAE WORKAREA WAS PROVIDED THEN
02       LOAD WORKAREA SIZE,
02       LOAD WORKAREA ADDRESS AND
02       FREE STAE WORKAREA
01     ENDIF
01     SAVE COMPLETION CODE FOR ABEND
01     LOAD SAVEAREA ADDR FOR FREE
01     LOAD PL/I-FORT SAVEAREA
01     LOAD A(PL/I PICA)
01     LOAD SAVEAREA SIZE FOR FREEMAIN
01     FREE THE SAVEAREA
01     SETUP FOR RESTORING PL/I SPIE
01     RESTORE THE PL/I SPIE EXIT
01     COMPLETION CODE TO REG. 1
01     SAVEAREA ADDRESS TO REG. 13
01     RESTORE PL/I REGS 2 TO 12
01     GO TO PL/I STAE EXIT
     ENDSEG DPIFRCVY
     THE FOLLOWING CODE HANDLES PL/I PATCH REQUESTS
     BGNSEG
01     A(PROBL)
01     SETUP BASE TO SUPL
01     IF NEEDED - SETUP NEW PROBL
01     ISSUE PATCH
01     STORE A(TCB EXTENSION)
01     CLEAR FLAGS FROM PL/I HALF-WORD
     ENDSEG DPPIFGO
     BGNSEG
01     MOVE FLAGS TO SUP LIST FLAGS
01     IF CORE MUST BE OBTAINED
02       SETUP FOR GETMAIN
02       GET THE CORE
02       COPY A(NEW PROBL)
02       A(NEW PROBL FOR MOVE)
02       LENGTH OF MOVE
02       COPY A(OLD PROBL)
02       LENGTH OF MOVE
02       MOVE PROBL
02       TELL SUPERVISOR TO FREE PROBL
```

Figure 3-79 (3 of 5). DPPPIF

```
01    ELSE
02      RETURN A(OLD PRCBL)
01    ENDIF
   ENDSEG GETPROBL
   THE FCLLCWING CODE HANDLES PL/1 PTIME REQUESTS
   BGNSEG
01    IF *: TYPE = RETURN TIME
02      GET CURRENT TIME
02      STORE TIME AND A(TIME ARRAY)
02      ZERC RETURN CODE
01    ELSE
02      A(PTIME PARAMETER LIST)
02      MCVE START FLAG
02      MOVE PURGE FLAG
02      MCVE STOP FLAG
02      A(PATCH AND PRCBL)
02      IF NEEDED - SETUP NEW PROBL
02      A(PROBL) BEING PASSED
02      SETUP PARAMETER REGISTERS
02      FCR CALLING PTIME
02      ISSUE
02      RESTORE
02      CLEAR FLAGS
02      FRCM PL/1
02      FIXED PCINT
02      FIELDS
01    ENDIF
   ENDSEG DPPIF04
   THE FCLLOWING CODE HANDLES PL/1 DPATCH REQUESTS
   BGNSEG
01    LOAD PURGE OPTICN
01    A(TASK NAME)
01    ISSUE DPATCH SVC
   ENDSEG DPPIF08
   THE FCLLOWING CODE HANDLES PL/1 REPATCH REQUESTS
   BGNSEG
01    LOAC A(REPL)
01    IF REC. TYPE IS ZERO, THEN
02      MOVE REPL TO USER CORE
02      RELOCATE SUPL FLAGS
02      MAKE QUEUE LENGTH A HALFWORD
02      AND COMPUTE ITS ADDRESS AND
02      REPLACE A(REPL)
02      SETUP A ZERO RETURN CODE
01    ELSE
02      RESTORE SUPL FLAGS
02      CLEAR TEMP. FLAGS LOCATION
02      COMPUTE REPL (RETRY/PURGE) OPTION
02      EXECUTE REPATCH SVC
01    ENDIF
```

Figure 3-79 (4 of 5). DPPPIF

```
   ENDSEG DPPIF12
   THE FOLLOWING CODE HANDLES PL/1 GET/PUTARRAY REQUESTS
   BGNSEG
01   LOAD A(NAME LIST)
01   INSERT NAME INCREMENT
01   LOAD A(DATA LIST)
01   INSERT DATA INCREMENT
01   CALL SERVICE
   ENDSEG DPPIF16
   THE FOLLOWING CODE HANDLES PL/1 GET/PUTITEM REQUESTS
   BGNSEG
01   LOAD A(NAME LIST)
01   INSERT NAME INCREMENT
01   LOAD A(DATA LIST)
01   INSERT DATA INCREMENT
01   CALL SERVICE
   ENDSEG DPPIF20
   THE FOLLOWING CODE HANDLES PL/1 GET/PUTBLOCK REQUESTS
   BGNSEG
01   LOAD A(DATA LIST)
01   PICKUP DATALIST INCREMENT
01   LOAD A(ARRAY NAME LIST)
01   CALL SERVICE
   ENDSEG DPPIF24
   THE FOLLOWING CODE HANDLES PL/1 MESSAGE REQUESTS
   BGNSEG
01   BUILD FIRST WORD OF MESSAGE
01   PARAMETERS TO CONTAIN
01   NO. VAR ADRS/ROUTES AND MSG NO.
01   PLACE ACTION CODE
01   LOAD STARTING ADDR. FOR SEARCH
01   LOAD INCREMENT FOR FULLWORDS
01   LOAD END OF SEARCH ADDRESS
01   STRTSRCH UNTIL END OF VARIABLES DO
02     TEST VARIABLE ADDRESS FOR ZERO
01   EXITIF A ZERO VARIABLE ADDRESS IS FOUND
02     LOAD START ADDRESS
02     COMPUTE NO. OF BYTES SEARCHED
02     COMPUTE NO. OF WORDS SEARCHED
02     RESET NO. OF VARIABLES COUNT
01   ENDLOOP
01   ENDSRCH
01   INSERT THE WAIT FLAG
01   A(MESSAGE PARM LIST)
01   CALL SUPPORT ROUTINE
01   CLEAR WAIT FLAG
01   CLEAR ACTION CODE
   ENDSEG DPPIF40
```

Figure 3-79 (5 of 5). DPPPIF

```
   THE FOLLOWING CODE HANDLES PL/1 PUTLOG REQUESTS
   BGNSEG
01   LOAD A(ARRAY NAME/NUMBER/LIST)
01   LOAD A(LOGHEADER/BLOCKLIST)
01   IF REG 0 CONTAINS A(BLOCKLIST)
02     LOAD INCREMENT VALUE
01   ENDIF
01   CALL SERVICE
   ENDSEG DPPIF44
   THE FOLLOWING CODE HANDLES PL/1 GETLOG REQUESTS
   BGNSEG
01   IF ITS AN ARRAY NAME
02     MOVE NAME ADDR TO LIST
01   ENDIF
01   LOAD A(GETLOG PARAMETER LIST)
01   CALL SERVICE
01   IF ITS AN ARRAY NAME
02     CLEAR ADDRESS FROM LIST
01   ENDIF
   ENDSEG DPPIF48
   THE FOLLOWING CODE HANDLES PL/1 DUMPLOG REQUESTS
   BGNSEG
01   IF A LIST OF NAMES OR NUMBERS OR
01   IF A NAME IS SPECIFIED
02     MOVE ADDR TO LIST
01   ENDIF
01   LOAD A(DUMPLOG PARAMETER LIST)
01   CALL SERVICE
01   IF A LIST ADDR
02     CLEAR ADDRESS FROM LIST
01   ENDIF
   ENDSEG DPPIF52
   THE FOLLOWING CODE HANDLES PL/1 RECORD REQUESTS
   BGNSEG
01   PICKUP LENGTH
01   INCLUDE ID
01   A(DATA)
01   CALL SUPPORT ROUTINE
   ENDSEG DPPIF56
   THE FOLLOWING CODE HANDLES PL/1 PATCH WAIT REQUESTS
   BGNSEG
01   A(ECB TO BE POSTED)
01   WAIT FOR A SINGLE EVENT
01   ZERO THE REGISTER
01   LOAD POST FLAGS AS RETURN CODE
01   CLEAR POST FLAGS FROM COMP. CODE
   ENDSEG DPPIF60
```

Figure 3-80. DPPSAMP1

```
      DPPSAMP1 MAIN SEGMENT 'SAMPLE PROGRAM PATCH ENTRY ROUTINE'
01       ISSUE SYSTEM MESSAGE 28
      ENDSEGMENT DPFSAMP1
```

Figure 3-81. DPPSASOC

```
' DDS ASYNCHRCNIS OPEN OR CLOSE'
GET ADDRESS OF INPUTS
INPUTS = A(IOA),WHICH CONTAINS OPEN/CLOSE SVC & PARMS
FUNCTICN = ASYNCHRCNOUSLY OPEN/CLOSE HALF A DDS
GET ADDRESS OF PARM FOR OPEN/CLOSE
EXECUTE OPEN/CLOSE SVC
POST THE ASYNCH ECB
RETURN TO CALLER
```

Figure 3-82 (1 of 2). DPPSBFST

```
   ' BLDL/FIND(TYPE-D/STOW FOR A DCS '
   INPUTS = (A(UDCB),A(PARAM),(TYPE))
   IF THIS DDS IS NOT DUPLICATED, THEN
01   CALL THE APPROPRIATE SEGMENT FOR USER'S OS DCB
   ELSE
01   IF THIS IS READCNLY, AND
01   IF STOW IS REQUESTED,THEN
02     SET RETURN CODE TO 32 AND RETURN TO USER
01   ENDIF
01   SHARE THIS DDS
01   CALL THE APPROPRIATE SEGMENT FOR THE PRIMARY CDSDCB
01   IF AN ERRCR CN THE PRIMARY, THEN
02     IF BACKUP IS IN-SERVICE, THEN
03       LOCK THIS DCS
03       SWITCH PRIMARY WITH BACKUP
03       UNLOCK & SHARE THIS CDS
03       IF THE SWITCHCVER WAS GOOD, THEN
04         EXECUTE SVC FOR THE NEW PRIMARY
03       ENDIF
02     ENDIF
01   ELSE
```

Figure 3-82 (2 of 2). DPPSBFST

```
02        IF IT IS NOT BLDL, AND
02        IF BACKUP IS IN SERVICE
03          PERFORM SVC ON BACKUP DCB
03          IF BACKUP FAILED, THEN
04            TAKE BACKUP OUT-OF-SERVICE
03          ENDIF
02        ENDIF
01      ENDIF
01      UNSHARE THIS DDS
    ENDIF
    RETURN TO THE CALLER
    BGNSEG
01      ISSUE A BLDL
01      IF THE BLDL FAILED, THEN
02        SET PROPER RETURN CODE
01      ENDIF
    ENDSEG BLDL
    BGNSEG
01      ISSUE FIND (TYPE D )
01      IF FIND FAILED, THEN
02        SET PROPER RETURN CODE
01      ENDIF
    ENDSEG FIND
    BGNSEG
01      ISSUE STOW ( TYPE A )
01      IF STOW FAILED, THEN
02        SET PROPER RETURN CODE
01      ENDIF
    ENDSEG STOWA
    BGNSEG
01      ISSUE STOW ( TYPE R )
01      IF STOW FAILED, THEN
02        SET PROPER RETURN CODE
01      ENDIF
    ENDSEG STOWR
    BGNSEG
01      ISSUE STOW (TYPE D )
01      IF STOW FAILED, THEN
02        SET PROPER RETURN CODE
01      ENDIF
    ENDSEG STOWD
    BGNSEG
01      ISSUE STOW ( TYPE C )
01      IF STOW FAILED, THEN
02        SET PROPER RETURN CODE
01      ENDIF
    ENDSEG STOWC
```

Figure 3-83. DPPSBF1

```
    ' BLDL/FIND(TYPE-D) FOR A DDS'
    INPUTS = $1=A(UCCB), $0=A(PARAM), $13=A(SAVE-AREA)
    IF FIND (TYPE-D), THEN
01    RECCMPLEMENT UDCB POINTER
    ENDIF
    CALL INTERNAL BUILDL/FIND/STOW ROUTINE - DPPSBFST
    RETURN TO CALLER
```

Figure 3-84 (1 of 2). DPPSCHCK

```
    ' DDS CHECK MODULE'

    INPUTS:  A(USER'S DECB) - DECCCBAC FIELD POINTS TO
    RESERVED DDSDECB


    SEE IF USER'S DECB AND DDSDECB POINT TO EACH OTHER

    IF THE UDECB IS CCNNECTED TO THE
    DDSDECB, AND
    IF THIS DDSDECB IS RESERVED,
01    THEN
01
01    DDSSHARE THIS DDS
01
01
01    SEE IF THIS DDSDECB IS ATTATCHED TO THE
01    PRIMARY CCB
01
01    IF THIS DDSDECB IS ATTATCHED TO
02      THE PRIMARY CCB, THEN
02
02      CALL DPPSCHK2 TO PERFORM ACTUAL CHECK
02      RETURN PARM 1 PCINTS TO UPCATED PRIMARY DECB
02      RETURN PARM 15 INDICATES NORMAL(3), EOC(1), OR SYNAD(0)
02
02
02
02      MOVE PRIMARY DECB TO USER'S DECB AREA
02
02      IF DSORG=DA
03        MCVE IN BCAM DECB
02      ELSE
03        MCVE IN BSAM DECB
02      ENDIF
02      IF OPENED FOR UPCATE, AND
02      IF THIS IS A READ, THEN
03        GET ADCRESS OF BACKUP IOB
02      ENDIF
```

Figure 3-84 (2 of 2). DPPSCHCK

```
01    ELSE
02      ZERO RETURN CODE
01    ENDIF
01
01    RELEASE THIS CDSDECB
01
01
01    DDSUNSHARE THIS DCS
01
    ELSE
01    ZERO RETURN CODE
    ENDIF

    RETURN AS PER INDICATED IN RETURN CODE TYPE-SYNAD,EOD, OR
    NCRMAL

    IF EOD OR SYNAD TAKEN
01    IF ECD TAKEN
02      SET RETURN ADDRESS FOR EOD
01    ELSE
02      SET RETURN ADDRESS FOR SYNAC
01    ENDIF
    ENDIF
    RETURN TO CALLER
```

Figure 3-85. DPPSCHK2

```
     ' CHECK A CDSDECB - INTERNAL '

     INPUTS = ( A(UCECB), A(PRIMARY DECB), A(IOA), A(CTLA),
     A(BACKUP DECB))
     IF I/O WAS STARTED, THEN

01     SET SYNAD & EOD FCR CHECK
01
01
01     BRANCH TO OS/VS CHECK RCUTINE
01
01
01     SEE IF SYNAD WAS TAKEN
01
01     IF SYNAD WAS TAKEN
01
02       THEN CALL THE SYNAD UPDATE CHECK RCUTINE
02       RETURNS ACDRESS OF PRIMARY CECB IN $1 & RETURN CODE IN $15
02
01     ELSE
01
02       SEE IF NCRMAL RETURN - I.E., NO EOD
02
02       IF NCRMAL RETURN
02       IF BACKUP IS IN-SERVICE, THEN
02
03         THEN CALL  CHECK UPDATE ON BACKUP
03
02       ENDIF
01     ENDIF
     ELSE
01     SET RETURN COCE TO INDICATE SYNAD
     ENDIF
     RETURN TO CALLER
```

Figure 3-86. DPPSCHK3

```
   ' SYNAD RCUTINE FOR DDS CHECK'
   INPUTS =(A(PRIMARY DECB), A(IOA), A(CTLA), A(BACKUP DECB))


   IF BACKUP IS IN-SERVICE

   THEN  SEE IF ERROR WAS A DISK FAILURE

   IF ERFCR IS A UNIT CHECK, OR
   IF ERROR IS A CFANNEL DATA CHECK,
01   A CHANNEL CONTROL CFECK, CR
01   AN INTERFACE CCNTFCL CHECK,
01   THEN
01   IF ERROR IS BUS CUT PARITY,
01   EQUIPMENT CHECK,
01   DATA CHECK, OR
01   OVERRUN, OR
01   IF ERROR IS PERMANENT ERROR, THEN
01
02     NCT A USER ERRCR SO SWITCH CVER IS CALLED FOR.
02
02
02     UNSFARE TFE DDS
02     LCCK TFE DDS
02
02
02
02
02     SWITCH PRIMARY WITH BACKUP
02
02     UNLOCK THIS DDS
02
02     SFARE TFE DCS
02
02
02     IF I/O STARTEC FOR NEW PRIMARY
03       THEN CHECK DECB(PRI)
03
03       SET SYNAD & EOD ADCRESSES
03
03
03       BRANCH TO OS/VS CHECK ROUTINE
03
02     ENDIF
01   ENDIF
   ENDIF
   RETURN TO CALLER
```

Figure 3-87. DPPSCHK4

```
   INPUTS = (A(BACKUP DECB),A(IOA),A(CTLA))
   'CHECK A BACKUP DECB'
   IF I/O WAS STARTED, THEN
01   CALL CS/VS CHECK ROUTINE
01   IF ECD OR SYNAD TAKEN, THEN
02     TAKE BACKUP OUT OF SERVICE
01   ENDIF
   ENDIF
   RETURN TO CALLER
```

Figure 3-88. DPPSCHPR

```
   'SET A(PRIMARY CECB) AND A(BACKUP DECB)'

   INPUTS =  ( A(CCSCECB),A(PRIMARY CCB))

   OUTPUTS =  $15 =  A(PRIMARY DECB) OR ZERO
   $0  =  A(BACKUP DECB) OR MEANINGLESS

   IF DECB1 IS PRIMARY
01   SET BACKUP ACCRESS TO CECB2
   ELSE
01   IF DECB2 IS PRIMARY
02     SET BACKUP ACCRESS TC DECB1
01   ELSE
02     INDICATE NO PRIMARY CECB
01   ENDIF
   ENDIF
```

Figure 3-89. DPPSCLUP

```
   'DDS CLEAN UP ROUTINE'
   INPUTS:  REGISTER       CONTENTS
   0     -A(TCBX)  IF ENTERED FROM DPPTPMON
   -A(TCB) IF ENTERED FROM PORJECTED O/S ETXR ROUTINE
   1      A(XCVT)
   2-12  NOT EXAMINED, SAVED & RETURNED
   13     CALLER'S SAVE AREA
   14     CALLER'S RETURN ADDRESS
   15     THIS ROUTINES ENTRY POINT
   FUNCTION:  CLEAN UP THE DDS FUNCTIONS LEFT OUTSTANDING
   BY THIS TASK

   GET 1ST TCBC
   STRTSRCH
   EXITIF
   ORELSE
   ENDLOOP
   ENDSRCH
   IF THIS TASK HAS A DDSLOCK, THEN
01   RELEASE THE DDSLOCK
   ELSE
01   IF THIS TASK SHARES A DDS
02     UNSHARE THAT DDS
01   ENDIF
   ENDIF
   WHILE THERE ARE MORE DDSDECBS T-CHAINED
01   REMOVE THESE DDSDECBS FROM THEIR IOA CHAINS
01   REMOVE THESE DDSDECB'S FROM THEIR T-CHAIN
   ENDDO

   WHILE THERE ARE MORE IOA'S FOR THIS TASK
01   PROCESS TERMINATION OF THESE IOA'S
01   WHILE THERE ARE MORE RSRVD DDSDECBS
02     FOR THIS IOA
02     REMOVE THESE DDSDECBS FROM THE T-CHAIN OF THE OWNER TASK
01   ENDDO
01   DISCONNECT THIS IOA FROM THE CTLA
01   RESTORE USER'S DCB TO PRE-DDSOPEN STATUS
01   FREE THE CORE FOR THIS IOA
   ENDDO
   CLEAR THIS TASKS IOA CHAIN ORIGIN
```

Figure 3-90. DPPSCL1

```
    ' CLOSE A DDS '
    INPUTS    $1 = (A(UDCB))
    SAVE INPUTS
    GET ACCRESS OF USERS DCB
    SEE IF THIS IS A DDS
    IF NOT A CDS
01    RESTORE REG1
01    ISSUE CLCSE SVC
    ELSE THIS IS A CDS
01    GET A(DDSICA)
01    IF PRIMARY CCB OPENED, THEN
02      CLCSE CCB1
01    ENDIF
01    IF BACKUP DCB IS OPENED, THEN
02      CLOSE CCB2
01    ENDIF
01    PREPARE TO RESTORE UDCB TO PRE/OPEN
01    STATLS
01    RESTORE UDCB BUFCB WORD
01    RESOTRE UCCB OFLAGS WORD
01    RESTORE UDCB CPTICN CODE WORC
01    RESTORE UDCB NCTE/POINT/CNTL WORD
01    STATUS
01    UNTIL
02      ALL DDSDECBS ARE CHECKEC
02      IF THIS CDSDECB RESERVD,THEN
02      ENDIF
01    ENDDO
01    FREE UP THE DCSIOA SPACE
    ENDIF
    RETURN TO CALLER
```

Figure 3-91 (1 of 2). DPPSCMPR

```
    ' COMPARE FOR DCS'
    FUNCTION - TO INVOKE THE IEBCOMPR MODULE TO PERFORM
    A COMPARE ON A CDS
    INPUTS - $1 POINTS TO A(CTLA), A(DDNAME1), A(DDNAME2)
    OUTPUTS - MESSAGES INDICATING COMPARE STATUS
    ROUTINES CALLED - LINKS TO IEBCOMPR
    RETURN CODE - NOT SET
    PREREQUISITES - REQUIRES A DDSCMPIN DD CARD IN THE JCL
    TO HOLD IEBCOMPR'S INPUT

    MOVE THE DDNAMES FROM THE INPUT DSECT TO THE DDNAME LIST
    FOR IEBCOMPR AND TO THE RDJFCB DCBS


    ESTABLISH THE ADDRESSES FOR THE EXIT LIST AND JFCB AREAS
    IN THE RDJFCB DCBS


    READ THE JFCBS



    IF
01    THE RETURN CODE IS NZERO, EXIT WITH THE ERROR MESSAGE:
01    'UNABLE TO ACCESS DATASETS'
    ENDIF

    MOVE THE DSORG (PS OR PO) TO THE TYPORG CONTROL CARD FOR
    IEBCOMPR, OPEN DDSCMPIN, PUT THE CONTROL CARD RECORD, AND
    CLOSE AND FREEPOOL DDSCMPIN
    NOTE: IF THE TYPORG'S ARE NOT EQUAL EXIT WITH
    THE ERROR MESSAGE:
    'DATA SETS NOT THE SAME TYPE'

    IF THE DDSCMPIN DCB CANNOT BE OPENED EXIT
    WITH THE ERROR MESSAGE:
    ' NO DDSCMPIN DD CARD'

    IF BPAM
01    SET TYPORG TO PO
    ELSE
01    SET TYPEORG TO PS
    ENDIF
    IF DSORGS NOT EQUAL
01    EXTT WITH ERROR MESSAGE
    ENDIF
    IF NO DDSCMPIN DD CARD
01    EXIT WITH ERROR MESSAGE
    ENDIF
```

Figure 3-91 (2 of 2). DPPSCMPR

```
OUTPUT MESSAGE: 'DDS COMPARE IN PROGRESS'


LOCK THE CDS


LINK TO THE IEBCOMPR MODULE

   SAVE THE RETURN CCDE AND UNLOCK THE DDS


   EXAMINE THE IEBCCMPR RETURN CODE AND EXIT WITH THE APPROPRIATE
   MESSAGE:
   RC=0 - 'CCMPARE ENDED  DATASETS EQUAL'
   RC NE 0 - 'CCMPARE ENDED - CATASETS NOT ECUAL'

   IF *: THE CCNCODE IS ZERO, THEN
01   PRINT CCMPARE ECUAL MSSGE
   ELSE
01   IF *: IF CCNCODE GREATER THAN 8, THEN
02     PRINT STATLS UKNOWN
01   ELSE
02     PRINT UNEQUAL MESSAGE
01   ENDIF
   ENDIF
```

Figure 3-92. DPPSCP2B

```
   ' COPY A DDS PRIMARY TO BACKUP'

   INPUTS =  'FROM' START, 'FROM' STOP, 'TO' START,
   A(OPENED INPUT DCB),
   A(OPENED OUTPUT DCB),A(DDNAME))


   GET A BUFFER LARGE ENOUGH TO HOLD A COMPLETE TRACK, 13600 BYTES


   GET THE 1ST FROM & TO TRACKS


   STRTSRCH THERE ARE NO MORE TKS

01   COPY TRACK TO TRACK:  PARMS = FROM CCHH, FROM DCB, TO CCHH,
01   TO DCB, A(BUFFER)
01
01
01   EXIT IF COPY WAS UNSUCCESFUL
01
   EXITIF COPY WAS NOT SUCCESSFUL
   ORELSE

01   GET NEXT FROM & TO TRACK
01
   ENDLOOP

01   SET THE RETURN CODE TO ZERO
01
   ENDSRCH

   FREE THE TRACK BUFFER


   RETURN TO CALLER
   BGNSEG $14 HAS CCHH TO BE INCREMENTED
01   IF NEED TO GO TO NEXT CYLINDER
02     GO TO NEXT CYLINDER
01   ENDIF
   ENDSEG BMPTRK   $14 HAS INCREMENTED CCHH
```

Figure 3-93. DPPSCRBK

```
    'CREATE A DDS BACKUP'
    INPUTS = (A(CTLA))
    IF BACKUP IS CUT-OF-SERVICE.
01    UPDATE THE BACKUP DSCB
01    SET A(ARJFCB) IN PRIMARY CCB
01    SET A(JFCB) IN ARJFCB
01    SET A(JFCB) IN CAMLIST AS DSNAME
01    SET A(VOL SER NO WITHIN JFCB) IN CAMLIST
01    SET A(DSCB) IN CAMLIST
01    MOVE IN PRIMARY DDNAME
01    MOVE IN BACKUP CCNAME
01    READ THE JFCB
01    IF JFCB READ IN O. K.
02      GET THE PRIMARY DSCB
02      MOVE IN DSORG
02      MCVE IN RECFM
02      MCVE IN CPTCD
02      MOVE IN KEYLEN
02      MCVE IN BLOCK SIZE
02      MOVE IN RECORD LENGTH
02      OPEN AND CLCSE FOR OUTPUT THE BACKUP
02      DCB SO AS TC UPDATE THE DSCB
02      IF
03        SET MACRF TO WI
02      ENDIF
01    ENDIF
01    CALL DPPSCP2B TO CCPY PRIMARY TO BACKUP
01    IF THIS DDS IS NCT OPEN, CR
01    IF IT IS NOT OUTPUT, THEN
02      GET THE ADDRESS OF THE VOLID OF PRIMARY IN $9
02      MOVE THE CCHHR OF THE PRIMARY DSCB TO THIS PROGRAM'S WORK AREA
02
02      CALL  DPPSDSCB TO UPDATE THE BACKUP DSCB TO HAVE THE
02      CURRENT TTRLL IN DS1LSTAR  (LAST RECORD PCINTER AND NO. BYTES
02      LEFT ON THIS TRACK)
01    ENDIF
01    IF COPY FAILED, THEN
02      SET RETURN CCDE AS SUCH
01    ELSE
02      IF THERE IS AN ICA, THEN
03        CALL DPPSCPCL TO ASYNCHRONOUSLY OPEN THE BACKUP DCB
03        IF DCB NCT OPENED FOR INPUT,
04          CALL DPPSRCIC TO RECREATE THE I/O ON THE BACKUP
04          IF BPAM DATASET
05            SET BACKUP DCB'S MEMBER TTR TO PRIMARY'S
04          ENDIF
03        ENDIF
02      ENDIF
01    ENDIF
    ELSE
01    UNABLE TO CCMPLY, PRINT BACKUP IS IN-SERVICE
    ENDIF
```

Figure 3-94 (1 of 4). DPPSCT2T

```
   ' COPY TRACK TO TRACK '
   INPUTS= 'FRCM' CCHH, 'TO' CCHH, INPUT DCB(OPENED),
   OUTPUT DCB(OPENED), A(BUFFER) TO HOLD TRACK

   *              CCMPLETE THE FIELDS IN THE 'FRCM' IOB              ******


   *              SET DATA ADDRESS IN EACH CCW OF THE CHANNEL PROGRAM ****

   UNTIL THE LAST CCW IS SET
01   SET ACDRESS IN CCW
   ENCDO

   *              CLEAR THE ECB                                      ******


   *              READ INTO CORE THE COUNT FIELDS FOR ALL RECORDS    ******
   *              (EXECUTE 'FROM' IOB CHANNEL PROGRAM)


   *              WAIT ON THE ECB                                       ***

   *              SEE IF THE CHANNEL PROGRAM TERMINATED WITH 'RECORD
   *              NOT FOUND'

   IF COMPLETICN CODE IS UNSUCCES
   IF SENSE BYTES INDICATE RNF
   IF UNIT CHECK IN CSW

01   *              SEE IF RECORD 0 WAS READ IN; IE; IS 1ST CCW COMPLETE
01
01   IF RECORD RO WAS READ INTC CORE
01
02     *              TURN OFF CCMMAND CHAINING FLAG FOR CCW PRIOR TO
02     *              FAILING CCW
02
02
02     *              CHANGE ALL CHAINED READ COUNT COMMANDS TO READ
02     *              CCUNT, KEY, & CATA COMMANDS
02
02
02     CALCULATE FOR ALL CHAINED READ COUNT CCWS THE COUNT
02     VALUES AND DATA ADDRESS VALUES FROM THE CORRESPONDING
02     COUNT FIELDS READ INTO CORE.
02
02     WHILE THE CCW IS CHAINED
03       SET NEW COMMAND CODE & DATA ACDRESS
03       CALCULATE NEW DATA ADDRESS FOR NEXT CCW
03       SET COUNT FOR THIS CCW
03       GET NEXT CCW
02     ENDDO
```

Figure 3-94 (2 of 4). DPPSCT2T

```
02
02
02
02      STRTSRCH NEXT CCW ISN'T CHAINED
02
03        *              CLEAR THE ECB
03
03
03        *              REBUILC THE IOB
03
03
03        *              READ IN ALL CATA FROM TRACK UP TO EOD; IE; EXECUTE
03        *              CHANNEL PROGRAM
03
03
03        *              WAIT ON THE ECB
03
03
03        *              SEE IF CHANNEL PROGRAM TERMINATED WITH EOD.
03        *              IF NOT, EXIT THE SEARCH
03
02      EXITIF NOT UNIT EXCEPTICN
03        IF NOT PERMANENT ERROR
03        GET CURRENT CCW ADDRESS
03        IF NOT EOD - POSITIVE DATA LENGTH
03
04          *              SEE IF CHANNEL PRCGRAM COMPLETED SUCCESSFULLY
04
04          IF NCT, ERROR EXIT
03        ORELSE
04          *              CHANNEL PROGRAM ENDED CN AN EOD
04
04
04          *              SET ADCRESS OF NEXT CCW IN DATA FIELD OF 2ND TIC CC
04
04
04          *              SET ADCRESS OF CCUNT DATA IN DATA FIELD OF SEARCH I
04
04
04          *              RESET START POINTER FOR IOB REBUILC TO SEARCH ID CC
04
04
04          SEE IF NEXT CCW IS CHAINED
04
03        ENDLOOP
03        ENDSRCH
03
```

Figure 3-94 (3 of 4). DPPSCT2T

```
03      *             COMPLETE TRACK SHOULD NCW BE IN BUFFER
03
03
03      CHANGE ALL CHAINED READ COUNT, KEY, & DATA CCWS
03      TO WRITE COUNT, KEY, & CATA,
03      *             AND SET THE 'TO' ACDRESS IN THE CYLINDER ADDRESS OF
03      *             EACH SUCH CCW.
03
03      WHILE THIS CCW IS CHAINEC
04        SET THE CCMMAND CODE TO WRITE CK&CATA
04        MOVE IN 'TO' CYLINDER ACDRESS
04        GET NEXT CCW
03      ENDDO
03
03      CHANGE THE CYLINDER ADDRESS OF RO CCUNT AREA
03      TO THE 'TO' ADDRESS
03
03
03      *             COMPLETE THE IOB FOR THE WRITE EXCP
03
03
03      SET UP THE WRITE CHANNEL PROGRAM AS FOLLOWS:
03
03      (1) SEARCH ID EQUAL ON THE COUNT FIELD OF RO.
03      (1.1) IF NO CATA IS TO BE WRITTEN
03      EXCEPT FOR RO, THEN SKIP TO (4)
03      (2) WHEN FOUND, TIC TC THE 1ST WRITE COUNT, KEY,
03      & DATA CCW, WHICH IS ALREADY CHAINED TO THE
03      NECESSARY STRING OF SUCH CCWS.
03      (3) CHAIN THE LAST CHAINED WRITE COUNT, KEY, &
03      CATA CCW TO SEARCH ID EQUAL ON THE COUNT FIELD
03      OF RO.
03      (4) WHEN THE SEARCH IS SATISFIED, TIC TO THE LAST
03      CCW, WHICH SHCULD BE A WRITE CATA FOR RC.
03
03      IF AT LEAST CNE
04        NON-RO DATA RECORD
04        IS TC BE WRITTEN, THEN
04        SET UP 2ND SEARCH ID CN RO
03      ELSE
04        GET A(WRITE RO) TO BE BUILT
04        SET THAT ADDRESS AS TIC ADDRESS
04        AFTER SEARCH ID CN RO IF GOOD
03      ENDIF
03
03      *             CLEAR THE ECB
03
03
```

Figure 3-94 (4 of 4). DPPSCT2T

```
03        *             WRITE OUT THE TRACK; IE; EXECUTE THE CHANNEL PROGRAM
03
03
03        *             WAIT CN THE ECB
03
03
03        *             SEE IF TFE CHANNEL PROGRAM CCMPLETED SUCCESSFULLY
03        *             IF NOT, ERROR EXIT
03
03
03        *             ZERC THE RETURN CODE
03
03
02      ELSE
03        UNABLE TO READ RECCRD 0 - SET RETURN CODE TO 12
02      ENCIF
01    ELSE
02      *             UNABLE TO READ THE COUNT FIELDS - SET RETURN CODE = 8
02
01    ENDIF
01    RETURN TO CALLER
```

Figure 3-95. DPPSDDSX

```
   'SEARCH FOR AN ICA FROM A GIVEN UCCB'
   INPUTS =(A(UDCB))
   GET START & STOP OF CTLAS
   STRTSRCH
01   WHILE THERE ARE STILL DDSCTLAS, DO
   EXITIF
01   IF CORRESPONCING IOA FOUND
01   ORELSE
02     GET NEXT CTLA
01   ENDLCOP
01   ENCSRCH
01   RETURN TO CALLER
```

Figure 3-96. DPPSINIT

```
   'INITIALIZE THE DDS SYSTEM'
   ADDRESS OF XCVT
   FILL CUT TEMPORARY         DR 846 *
   CTLA WITH                  DR 846 *
   DCNAMES AS PER CTLIN STREAM
   IF UNABLE TO OPEN           DR 846 *
   DDSTATUS, OR             DR 846 *
   SET FLAG FOR NOSTAT MESSAGE
   IF *: THERE ARE NO CECLARATICNS
01   SET END = START
   ELSE
01   IF ERROR IN CTLIN STREAM
02     ABEND WITH CODE = 80
01   ENDIF
01   DETERMINE THAT EACH DDS
01   DECLARATICN IS CORRECT
01   WRT JCL
01   IF BAD DDS DECLARATION
02     ABEND WITH CODE = 80
01   ENDIF
   ENDIF
   ALLOCATE DCSCTLA, MOVE
   TEMPORARY CTLA INTO IT.
   SET THE DDSCTLA ADDRESS
   IN THE SCVT
   IF ERROR IN DDSCTLA ALLOCATICN
01   ABEND WITH CODE = 80
   ENDIF
   CALCULATE MAX BLKSZE DR 846 *
   SET MAX BLOCKSIZE IN DDSCTLHD FOR WRST
   IF THIS IS READONLY       DR 846 *
01   MODE, THEN             DR 846 *
01   CR 846 *
01   TURN CN READCNLY BIT            DR 846 *
   ELSE DR 846 *
01   CALL TO WRITE DDSTATUS
   ENDIF DR 846 *
   IF NOSTAT MESSAGE REQUIRED, THEN
01   OUTPUT THE MESSAGE
   ENDIF
   RETURN TO CALLER
```

Figure 3-97 (1 of 2). DPPSINI2

```
     'PROCESS THE CTLIN STREAM'
     INPUTS = (A(CTLA),MAXDDS)
     OPEN THE INPUT STREAM
     IF *: INPUT STREAM SUPPLIED,
01     GET LENGTH OF EACH CTLA        DR 846 *
01     MULTIPLY BY MAX NO. CTLA'S TO GET        DR 846 *
01     MAX BLOCKSIZE FOR DDSTATUS               DR 846 *
01     SET IN DDSTATUS DCB            DR 846 *
01     GET THE 1ST CARD                 DR 846 *
01     GET 1ST PARM        DR 846 *
01     SET READONLY FLAG ON                DR 846 *
01     IF ',OR READONLY WAS    DR 846 *
01     SELECTED, OR    DR 846 *
01     SET READONLY BIT OFF
01     IF REFRESH WAS     DR 846 *
02       SELECTED, THEN DR 846 *
02       SET UP THE DDSCTLA'S AS PER DDSTATUS DATASET                       DR 846 *
02       OPEN DDSTATUS  DR 846 *
02       IF DDSTATUS OPENED    DR 846 *
03         OK, THEN            DR 846 *
03
03         CHECK THE READ                          DR 846 *
03         GET ADDRESS OF IOB                 DR 846 *
03         GET MAX BLOCK SIZE              DR 846 *
03         SUBTRACT BYTES NOT READ TO          DR 846 *
03         GET ACTUAL DDSTATUS RECORD          DR 846 *
03         LENGTH                              DR 846 *
03         SET ADDRESS OF END OF DDSCTLA'S   DR 846 *
03         CLOSE DDSTATUS          DR 846 *
03         CLOSE THE INPUT STREAM     DR 846 *
03         INDICATE NORMAL RETURN CODE           DR 846 *
03         SET RETURN REG 1 TO ADDRESS OF       DR 846 *
03         END OF CTLA'S                        DR 846 *
03         INDICATE REFRESH IN RETREG 0             DR 846 *
03         IF READONLY SPECIFIED,THEN DR 846 *
04           INDICATE READONLY IN RETREG 0            DR 846 *
03         ENDIF DR 846 *
02       ELSE DR 846 *
03         INDICATE UNABLE TO OPEN DDSTATUS       DR 846 *
02       ENDIF DR 846 *
01     ELSE DR 846 *
02       UNTIL *: MAXIMUM CHECKED
03         PROCESS THIS CARD
03         GET NEXT SLOT IN CTLA
02       ENDDO
01     ENDIF
```

Figure 3-97 (2 of 2). DPPSINI2

```
01    BGNSEG
02      CLCSE THE INPUT STREAM
02      ACCRESS OF END CF CTLA
02      INDICATE STANCARD START MODE           DR 846 *
01    ENDSEG CARDEOD
    ELSE
01    RETURN WITH CCNCCDE = 4
    ENDIF
    RETURN TO CALLER
    BGNSEG DR 846 *
01    THIS SEGMENT RETURNS THE PROPER RETURN CODE ON SYNAD OR EOD DR 846 *
01    OFF DDSTATUS READ                                           DR 846 *
01    CLOSE DDSTATUS
    ENDSEG SYNEOD                                           DR 846 *
```

Figure 3-98. DPPSINI3

```
    'VALICATE EACH CDS DECLARATION WRT JCL'
    INPUTS = (A(CTLA),A(CTLAEND)
    SET THE ADDRESSES FOR READ JFCBS
    WHILE THERE ARE MORE CTLA'S
01    MOVE IN DDNAME1
01    MOVE IN DCNAME2
01    REAC THE JFCBS FCR EACH DCB
01    ERROR IF NC CD CARD
01    INSERT OTHER JFCB VALIDITY CHECKS HERE
01    CLEAR THE JFCBS
01    GET NEXT CTLA ENTRY
    ENDDO
    DDS DECLARATICNS PASS VAL CHECK
    RETURN TO CALLER
```

Figure 3-99. DPPSINI4

```
    'ALLOCATE AND INITIALIZE PERMANENT CDSCTLA'
    GET ACCRESS OF SCVT
    CALCULATE LENGTH CF CTLA
    SET LENGTH OF DCSCTL
    FOR GETMAIN
    ALLOCATE CDSCTLA SPACE
    SAVE CCSCTLA START ADDRESS
    CLEAR CCSCTLHD IN ITS ENTIRETY
    GET ACCRESS OF 1ST ENTRY
    IN DDSCTL
    ADDRESS OF END OF DDSCTLA
    STORE THESE TWO ADCRESSES IN DDSCTLA HDR
    LCAD I/C ROUTINES & SAVE THEIR ADDRESSES
    GET ADDRESS OF DPPSNTPT
    GET DDS NOTE RCUTINE
    SAVE ADDRESS IN NTPT
    GET POINT/FINC(TYPE C) ROUTINE
    SAVE ADDRESS IN NTPT
    GET DDS SHARE ROUTINE
    SAVE ADDRESS IN CTLHD
    GET DDS UNSHARE RCUTINE
    SAVE ADDRESS IN CTLHD
    GET DDS LCCK ROUTINE
    SAVE ADCRESS IN CTLHD
    GET DDS UNLCCK RCUTINE
    SAVE ACCRESS IN CTLHD
    ADDRESS OF
    DDSCTLA IN
    SCVT
    RESET START CTLA ADDRESS
    IF
01    ANY CCS' WERE DECLARED, THEN
01    MOVE CTLA INTO DDSCTLA
    ENDIF
    RETURN TO CALLER
```

Figure 3-100. DPPSINI5

```
    'DEFORMAT THE CTLIN CARD'
    INPUTS=(A(CARD),A(END),A(CTLA))
    CLEAR THE DDNAMES
    ZERO REMAINDER OF CTLA ENTRY
    FIND 1ST NON-BLANK CHARACTER
    IF *: A USER DDNAME WAS ENTERED
01    GET THE USERS DDNAME
01    MOVE IN USER'S DDNAME
01    FIND NEXT NON-BLANK CHARACTER
    ENDIF
    START OF DDS PAIR NAMES
    GET NEXT NON-BLANK CHARACTER
    GET THE 1ST DDNAME
    MOVE IT TO THE CTLA
    IF ',THEN MOVE IT TO UDDN ALSO
    ENDIF
    GET 2ND DDNAME
    IF BACKUP IS OUT-OF-SERVICE
01    IF ',THEN
02      THE REQUEST IS FORMATTED WRONG, THEN
02      ERROR EXIT
01    ENDIF
01    SET BACKUP OUT-OF-SERVICE FLAG ON
    ELSE
01    ERROR EXIT IF NOT FORMATTED CORRECTLY
    ENDIF
    BGNSEG
01    SET RETURN REG TO ADDRESS OF NEXT DELIMETER
    ENDSEG FINDP
```

Figure 3-101. DPPSINI6

```
    'DEFINE LOCKS FOR EACH CTLA'

    *               INPUTS =  A(CTLHD),A(XCVT)

    WHILE THERE ARE MORE CTLAS
01    DEFINE A LOCK FOR THIS DDS & CLEAR THE LOCK/SHAR WORDS
    ENDDO
```

Figure 3-102. DPPSLOCK

```
    ' LCCK A CDS '
    GET ACDRESS OF.CTLA
    GET ACCRESS OF ECB
    CLEAR THIS LECB
    WHILE THERE ARE MORE LECBS
    BGNWHILE
01    GET NEXT LECB
    ENDDO
    PUT THIS LECB IN CHAIN
    IF THERE ARE SECBS, OR
    IF THERE IS A PRIOR LOCK
01    WAIT ON THIS ECB
    ELSE
01    LET CALLER HAVE CCNTROL NCW
    ENDIF
```

Figure 3-103 (1 of 3). DPPSMSGI

```
    'DDS INPUT MESSAGE PROCESSOR'
    INPUTS = (A(XCVT),A(RESTBL),A(PRCBL))
    PRCBL = ((LG,,ID),(L,A(1ST-PARM)),...,(L,A(LAST-PARM)))
    IF THERE IS A PARM 1, THEN
01    MOVE IN THE 1ST PARM
    ENDIF
    IF DSNAME CODED, THEN
01    SKIP CVER KEYWORD
    ENDIF
    IF THIS DDS NOT DECLARED, THEN
01    ERROR MESSAGE
    ELSE
01    IF A 2ND PARM WAS ENTERED
02      IF A 2ND PARM, THEN
03        MCVE IN THE 2ND PARM
02      ENDIF
01    ENDIF
01    IF ',THEN  NC PARM2 ENTERED
02      DEFAULT TO STATUS
01    ENDIF
01    IF REQUEST NCT UNDERSTCOD, THEN
02      ERROR MESSAGE - REQUEST NOT UNDERSTOOD
01    ELSE
02      PROCESS THE REQUEST
01    ENDIF
    ENDIF
    BGNSEG
```

Figure 3-103 (2 of 3). DPPSMSGI

```
01     TAKE THE BACKUP OUT-OF-SERVICE
     ENDSEG TAKE
     BGNSEG
01     ATTEMPT TO CREATE THE BACKUP
01     IF THIS IS READONLY MODE
02       OUTPUT NO COPY MESSAGE AND RETURN
01     ELSE
02       IF
03         THE CREATE WAS SUCCESSFUL, THEN
03         PRINT MESSAGE INDICATING SUCCESSFUL CREATE
03         UPDATE DDSTATUS RECORD
02       ELSE
03         IF BACKUP IN SRVICE,THEN
04           PRINT MESSAGE INDICATING BACKUP ALREADY IN-SERVICE
03         ELSE
04           PRINT MESSAGE INDICATING BAD CREATE
03         ENDIF
02       ENDIF
01     ENDIF
     ENDSEG CREATE
     BGNSEG
01     IF
02       THE BACKUP IS IN SERVICE, THEN
02       PRINT A MESSAGE FOR BOTH DATA SETS
01     ELSE
02       PRINT A MESSAGE FOR PRIMARY DATASET- BACKUP=OUT-OF-SERVICE
01     ENDIF
     ENDSEG STATUS
     BGNSEG
01     IF
02       THE BACKUP IS NOT IN SERVICE, THEN
02       PRINT A MESSAGE INDICATING SWITCH NOT POSSIBLE
01     ENDIF
     ENDSEG SWITCH
     BGNSEG
01     IF NO OPENED DDSCCB'S
02       SEE IF A THIRD DD NAME WAS SPECIFIED
02       STRTSRCH THREE BLANKS ENCOUNTERED
02       EXITIF NO REPLACEMENT SPECIFIED
02       ORELSE
03         CHECK NEXT PARM
02       ENDLOOP
03         IF
04           THE REPLACEMENT DDNAME EXCEEDS  8 CHARACTERS, THEN
04           LIMIT REPLACEMENT DDNAME SIZE TO 8 CHARACTERS
03         ENDIF
```

Figure 3-103 (3 of 3). DPPSMSGI

```
03          MCVE IN REPLACEMENT CCNAME
02        ENDSRCH
02        SET TFE NEW DCNAMES IN THE CTLA
02        UPDATE CCSTATUS RECORD
01      ELSE
02        PRINT A MESSAGE INDICATING REPLACE REQUEST IS PRECLUDED BY AN
02        OPEN DDSCCB
01      FNDIF
    ENDSEG REPLACE
    BGNSEG
01      IF
02        A PARAMETER CAN EXIST, THEN
02        SEARCH FOR A BLANK, WHICH GETS THE PARAMETER
01      ENDIF
    ENDSEG FINDP
    BGNSEG
01      IF ANOTHER PARM ENTERED
02        IF THIS PARM IS NOT NULL, AND
02        IF ',THEN  1ST CHARACTER NOT A BLANK
03          MCVE IN USER SPECIFIED DD1
02        ENDIF
02        IF ANCTHER PARM ENTERED
03          IF PARM IS NOT NULL
03          IF ',THEN 1ST CHARACTER NOT A BLANK
04            MOVE IN USER SPECIFIED DD2
03          ENDIF
02        ENDIF
01      ENDIF
01      LINK TO CCMPARE ROUTINE
    ENDSEG CCMPARE
```

Figure 3-104. DPPSMSGO

```
    'DDS MESSAGE OUTPUT PROCESSCR'
    INPUTS = (IC,A(PARM1),...,A(LAST PARM))  MAX=5
    IF
01      THE OUT MESSAGE PROCESSOR IS INITIALIZEC, THEN
01      ISSUE THE APPROPRIATE MESSAGE MACRO
    ENDIF
```

Figure 3-105. DPPSNOTE

```
'PERFCRM NOTE ON A CDS '
INPUTS  $1 = A(UDCB)
SAVE $1
GET ACCESS CF IOA
ISSUE NCTE CN PRIMARY DCB
```

Figure 3-106. DPPSNTPT

```
' NOTE OR PCINT FOR DDS'
CALL THE NOTE OR POINT ROUTINE FCR PROCESSING
```

Figure 3-107. DPPSOPCL

```
   ' OPEN/CLCSE HALF OF A DDS'
   INPUTS = (A(IOA))
   SAVE INPUTS
   GET ADDRESS CF IOB
   GET ACCRESS OF CVT
   GET ACCRESS OF CURRENT TCB
   GET ADDRESS OF CPENER TCB
   IF THIS IS CPENER TCB
01   EXECUTE THE OPEN/CLCSE SVC FRCM IOA
01   SET ZERO RETURN CCDE
   ELSE
01   SAVE RESUME PSW
01   VALICATE THAT OPENER TCB IS STILL DEFINED UNDER THIS JOB STEP
01   GET JOB STEP TCB
01   THIS VALICATICN ALGORITHM ASSUMES THAT
01   THE JCB STEP TASK PRIORITY IS GE
01   TO THE HIGHEST PRICRITY CF ALL CAUGHTER
01   TASKS
01   STRTSRCH THERE IS A LOWER TCB
01   EXITIF CPENER TCB FOUND
02     IF TCB NOT IN SAME JOB STEP
03       SET RTN CCDE AS BAD
02     ENDIF
01   ORELSE
02     GET NEXT LCWER TCB
01   ENDLOOP
01   ENDSRCH
01   IF OPENER TCB IS STILL VALID
02     GET ADDRESS OF DDS ASYNCHRCNYS
02     OPEN/CLCSE RCUTING
02     CREATE IRB FCR CLOSE
02     SAVE THE IRB ADDRESS
02     GET ACCRESS CF WORK AREA (IQE)
02     CLEAR THE ICE
02     PUT ACDRESS CF IOA IN ICE
02     PUT ACDRESS CF IRB IN ICE
02     PUT ADDRESS OF CPENER TCB IN IQE
02     CLEAR THE AECB
02     CALL STACE2 EXIT EFFECTOR
02     RESET THE PSW
02     WAIT ON ASYNCH ECB
02     SET ZERC RETURN CODE
01   ELSE
02     RESET THE PSW
02     SET RETURN CODE FOR UNABLE TO CCMPLY
01   ENDIF
   ENDIF
```

Figure 3-108. DPPSOP1

```
    ' CPEN A CDSCCB'

    INPUTS = A(CFEN PARAMETERS) = ((CPTICN BYTE),AL3(USER'S DCB))


    SEE IF THIS IS A CECLARED DDS


    IF THIS IS NCT A DECLARED CDS

01    PERFORM STANDARD CS/VS OPEN
01
    ELSE
01    IF THIS IS REACCNLY MODE, AND
01    IF OPEN FCR CUTPUT, THEN
02      RETURN TO USER WITHCUT CPEN
01    ENDIF
01
01    DDS LOCK THIS DDS
01
01    IF THIS CTLA HAS NCT YET BEEN
02      CONNECTED
02
02      CALL CP2 TC CCMPLETE THE CUAL OPEN
02
02      A(CPTICN BYTE),A(USER'S DCB),A(CTLA),A(CTLHD)
02
01    ENDIF
01
01    CDS UNLCCK THIS CCS
01
01
    ENDIF
```

Figure 3-109 (1 of 2). DPPSOP2

```
    'OPEN A DECLAREC CCS'

    INPUTS = A(CPTICN BYTE), A(USER'S DCB), A(CTLA), A(CTLHD)



    GET TFE MAIN CCRE FCR THE ICA TO BE BUILT

    IF NCP SHCULB BE 1
01    SET NCP VALUE TO 1
    ENDIF

    INITIALIZE TFE DDSICA

    UNTIL
01    THE ENTIRE AREA IS CLEARED
01    CLEAR THE NEXT 256 BYTE PORTION
    ENDDO
    IF STILL MORE BYTES TO CLEAR
01    CLEAR THOSE BYTES
    ENDIF
    IF THIS IS A BPAM DS, OR
    IF THIS I S A BSAM DS, AND
    IF THIS IS NCT CREATE BCAM,
01    FORCE THE PCINT CPTION
    ENDIF
    UNTIL ALL DDSDECBS ARE CONNECTED
01    CONNECT THIS CDSDECB
01    GET NEXT DDSDECB
    ENDDO

    CONNECT THE CTLA WITH THE IOA


    ATTEMPT TO OPEN PRIMARY CCB


    IF DCB1 OPENED OK

01    IF BACKUP IS IN SERVICE
01
02      ATTEMPT TO OPEN BACKUP DCB
02
02
02      IF DCB2 FAILED TO OPEN
02
```

Figure 3-109 (2 of 2). DPPSOP2

```
03       TAKE BACKUP CUT-OF-SERVICE
03
03
02     ENDIF
01   ENDIF
01
   ELSE
01   DCB1 FAILED TC CPEN
01
01   ERROR IF BACKUP
01   *: OUT-OF-SERVICE
01
01   SWITCH PRIMARY WITH BACKUP & TAKE BACKUP OUT-OF-SERVICE
01
01
01   ATTEMPT TO OPEN PRIMARY DCB
01
01
01   ERROR IF CCB
01   *: FAILED TO OPEN
   ENDIF

   CCNNECT USER'S CCB WITH THIS DDSIOA


   FREE TFE IOA CORE


   DISCONNECT THE CTLA FRCM THE IOA
```

Figure 3-110. DPPSPNTF

```
   ' PERFORM POINT/FINC(TYPE C) CN A DDS '
   INPLTS:  $1= A(CCB)  $0 = A(LIST) , $13 = A(SAVE AREA)
   SAVE PARAMO
   SAVE PARAM1
   GET CALLER'S SAVE AREA
   GET REUTRN ADDRESS
   POINT TO PREVIOUS INSTRUCTICN
   GET ACCRESS CF IOA
   IF POINT REQUESTED
01   PCINT FOR PRIMARY DCB
01   IF BACKUP IN SERVICE
02     POINT FCR BACKUP CCB
01   ENDIF
   ELSE TYPE C FIND RECUESTEC
01   FIND FOR THE PRIMARY
01   IF BACKUP IN SERVICE
02     FIND FOR THE BACKUP
01   ENDIF
   ENDIF
```

Figure 3-111. DPPSRCIO

```
   'RECREATE I/O FCR A DDS HALF '
   INPUTS = A(CTLA), A(IOA), A(FROM-DCB), A(TO-DCB)
   GET ADDRESS OF 1ST UNCHECKEC CDSDECB
   IF THERE IS AT LEAST ONE UNCHECKED
01   DDSDECB, THEN
01   IF DSORG IS BSAM CR BPAM, THEN
02     GET THE PRIMARY DECB
02
02     POINT 'TC-DCB' TC TTR OF 'FROM-DCB'S' 1ST UNCHECKED DECB'S IOB'S
02     SEEK ADDRESS
02
01   ENDIF
01   WHILE THERE ARE MORE DDSDECBS
01   BGNWHILE
02     PREPARE TO-CECB HALF FOR I/O AND CALL CS/VS READ/WRITE
02     GET THE NEXT UNCHECKED CDSCECB, IF ANY
01   ENDCO
   ELSE
01   IF CSORG IS BSAM CR BPAM, THEN
02     PCINT THE RECREATED DATASET AS PER OLD  DCB'S  DISK ADDRESS
01   ENDIF
   ENCIF
   BGNSEG
01   CALL THE OS ROUTINE TO CCNVERT MBBCCHRR TO TTR
   ENDSEG CNVRT
```

Figure 3-112. DPPSRDWT

```
    ' READ/WRITE MODULE FOR DDS'

    INPUTS =  A(USER'S DECB)

    GET THE ADDRESSES OF THE USER'S DCB, THE IOA, AND THE CTLA.

    GET A(DDSCTLHD)
    IF THIS IS READONLY, THEN
01    IF THIS IS BDAM, THEN
02      IF THIS IS A WRITE, THEN
03        RETURN TO USER WITHOUT DOING WRITE
02      ENDIF
01    ELSE
02      THIS IS BSAM/BPAM
02      IF THIS IS A WRITE, THEN
03        RETURN TO USER WITHOUT DOING WRITE
02      ENDIF
01    ENDIF
    ENDIF

    DDS SHARE THE DDS


    RESERVE AN AVAILABLE DDSDECB - EXIT IF NONE AVAILABLE



    IF A DDS WAS RESERVED, THEN

01    CALL RDW2 FOR ACTUAL READ/WRITE
01
    ELSE
01    SET CODE FOR NO DDSDECBS
    ENDIF

    DDS UNSHARE THIS DDS
```

Figure 3-113. DPPSRDW2

```
   'ACTUAL READ/WRITE FCR A DDS'

   INPUTS = (A(CDSDECB), A(IOA), A(CTLA), A(USER'S DECB))


   PREPARE DCSCECB1 FCR I/O


   BRANCH TO OS/VS REAC/WRITE ROUTINE


   IF NOT CPENED FOR INPUT, ANC
   IF BACKUP IS IN-SERVICE


01   PREPARE DCSCECB2 FOR I/O
01
01   IF CPEND FOR UPCATE, AND
01   IF THIS IS A WRITE, THEN
02     GET A(BACKUF ICB) FROM USER'S DECB
01   ENDIF
01
01   ERANCH TO OS/VS READ/WRITE RCUTINE
01
01
   ENDIF
```

Figure 3-114. DPPSRLSE

```
'RFLEASE A CDSDECB'
INPUTS = (A(IOA),A(DDSDECB))
DE-CHAIN THIS DCSDECB FRCM ITS TCBX-DDS-CHAIN
DE-CHAIN THIS DDSDECB FROM ITS DDSIOA CHAIN
```

Figure 3-115. DPPSRSRV

```
   'RESERVE A CDSDECB'
   INPUTS = (A(IOA),A(CTLA))
   RESERVE AN AVAILABLE DDSDECB
   STRTSRCH ALL CDSDECBS HAVE
01   BEEN  CHECKED
   EXITIF THIS DDSDECB IS AVAILABLE
01   CHAIN THIS CDSDECB TO ITS IOA AND ITS TCB-DDS-CHAIN
   ORELSE
01   GET NEXT DDSDECB
   ENDLOCP
01   INDICATE NO DDSDECB AVAILABLE
   ENDSRCH
```

Figure 3-116 (1 of 2). DPPSRSTR

```
   GET A(XCVT)
   GET A(SCVT)
   GET A(DDSCTLHD)
   IF THIS IS READCNLY MODE, AND
   IF THIS IS WRITE RESTART, THEN
01   RETURN TO TFE USER
   ENDIF
   TURN CFF REAC CNLY BIT
   CALL DPPSTKCK TO CHECK TASKS
   FOR DDS VIOLATICNS
   GET A(DCSCTLA'S)
   SET MAX DDSTATUS BLCCKSIZE
   OPEN DDSTATUS FOR INPUT
   IF OPEN FAILED, TFEN
01   OUTPUT NC OPEN(INPUT) MESSAGE
01   SET RETURN CODE TC 8 AND RETURN TO CALLER
01   CLOSE DDSTATUS
01   OUTPUT SYNAD ON RSTR
01   CLOSE DCSTATUS
01   UTPUT EODAC CN RSTR
   ENDIF
   READ TFE DDSTATUS RECORD ANC CHECK THE READ
   CALCULATE THE DDSTATUS RECORD LENGTH
   GET A(IOB)
   GET RESIDUAL COUNT
   GET MAX BLCCKSIZE
   GET ACTUAL NC. BYTES READ
   CALCULATE A(END OF CDSTATUS RECORD)
   CLOSE DDSTATUS
   WHILE THIS IS NCT END OF IN-CORE CTLA
```

Figure 3-116 (2 of 2). DPPSRSTR

```
01    SEARCH THROUGH DDSTATUS RECCRD FOR THIS ENTRY.
01    IF THIS CTLA IS IN DDSTATUS
02      UPCATE INCORE DDSCTLA
02      MCVE IN PRIMARY DDNAME
02      MOVE IN BACKUP DDNAME
02      IF DDSTATUS BACKUP IS IN-SERVICE,
03        THEN
03        SET BACKUP OUT-OF-SERVICE FLAG ON
03        IN IN-CORE CDSCTLA
02      ELSE
03        SET BACKUP OUT-OF-SERVICE FLAG OFF
02      ENDIF
02      UPDATE IN-CORE USED CCUNTER
01    ELSE
02      UPCATE IN-CORE FLAG COUNTER
01    ENDIF
01    GET NEXT INCORE DDSCTLA
    ENDDO
    IF MISSING DDSTATUS DDSNAME
01    OUTPUT 'SMALLER'
01    MESSAGE
    ENDIF
    CALCULATE # DDSNAMES IN DDSTATUS
    IF MISSING DDSNAMES IN CORE
01    OUTPUT 'LARGER'
01    MESSAGE
    ENDIF
    OUTPUT 'RESTART
    COMPLETED' MESSAGE
    CALL DPPSWRST TO WRITE THE NEW
    DDSTATUS RECORD
```

FIGURE 3-116.1. DPPSRTCP

```
    DPPSRTCP - REALTIME COPY
01  READ THE PRIMARY JFCB
01  IF JFCB READ WAS SUCCESSFUL
02      OBTAIN THE PRIMARY DSCB
02      OPEN PRIMARY DCB
02      OPEN SECONDARY DCB
02      CALL DPPSCP2B TO COPY PRIMARY TO BACKUP
02      CALL DPPSDSCB TO UPDATE THIS BACKUP DSCB
02      CLOSE PRIMARY DCB
02      CLOSE SECONDARY DCB
01  ENDIF
01  IF ERROR OCCURRED
02      SET RETURN CODE
02      ISSUE ERROR MESSAGE
01  ELSE
02      ISSUE COPY COMPLETED MESSAGE
01  ENDIF
```

Figure 3-117. DPPSSHAR

```
   'SHARE A DDS'
   GET ADDRESS OF CTLA
   GET ADDRESS OF ECB
   CLEAR THIS SECB
   IF THIS DDS IS LOCKED OUT
01   WHILE THERE ARE MORE SECBS
02     GET NEXT SECB
01   ENDDO
01   ADD THE SECB TO THE CHAIN
01   WAIT ON THIS ECB
   ELSE
01   GET SHARE COUNTER
01   INCREASE IT BY 1
01   PUT BACK IN SHARE COUNTER
   ENDIF
```

Figure 3-118. DPPSSRCH

```
   'SEARCH A FIXED LENGTH TABLE FOR AN ENTRY WHOSE        X
   KEY MATCHES THAT OF AN ARGUMENT',PS ECT=NO
   STRTSRCH
01   WHILE THERE ARE MORE ENTRIES IN THE TABLE
   EXITIF ENTRY FOUND
01   SET RETURN REGISTER TO CORRESPONDING ENTRIES ADDRESS
   ORELSE
01   GET NEXT ENTRY IN THE TABLE
   ENDLOOP
01   SET RETURN REGISTER TO ZERO, INDICATES NO ENTRY FOUND
   ENDSRCH
```

Figure 3-119. DPPSST1

```
   ' STOW FOR A DDS'
   INPUTS = $1=A(UCCB), $0=A(PARAM), $13=A(SAVE AREA)
   IF THIS IS TYPE-C, THEN
01   SET TYPE INDICATOR & RECOMPLIMENT BOTH INPUTS
   ELSE
01   IF THIS IS TYPE-D, THEN
02     SET TYPE INDICATOR & RECOMPLIMENT IN PARAM
01   ELSE
02     IF THIS IS TYPE-R, THEN
03       SET TYPE INDICATOR & RECOMPLIMENT A(UCCB)
02     ENDIF
01   ENDIF
   ENDIF
   CALL THE INTERNAL BLDL, FIND, STOW ROUTINE
```

Figure 3-120. DPPSSWCH

```
   'SWITCH PRIMARY-TO-BACKUP FOR A DDS'
   INPUTS =(A(CTLA))
   IF BACKUP IS IN-SERVICE, THEN
01   IF THERE IS AN IOA, THEN
02    ' IF THE CCSCCB WAS CPENED FOR
03        INPUT, THEN
03        RECREATE
03        THE I/O ON THE
03        BACKUP DCB
02      ENDIF
02      SWITCH ADDRESSES OF PFIMARY & BACKUP CCBS IN THE IOA
01   ENDIF
01   SWITCH CDNAMES IN THE CTLA
01   TAKE THE BACKUP CUT-OF-SERVICE
01   SET RETURN CODE TC ZERO
   ELSE
01   SET RETURN CODE TO 4
   ENDIF
```

Figure 3-121. DPPSTBOS

```
   'TAKE A BACKUP OUT-OF-SERVICE'


   INPUTS = (A(CTLA))

   SEE IF BACKUP IS ALREACY OUT-OF-SERVICE

   IF BACKUP IS OUT-OF-SERVICE
01   SET RETURN CODE TC INDICATE BACKUP-ALREADY OUT-OF-SERVICE
   ELSE


01   SET OUT-OF-SERVICE FLAG FOR BACKUP DCB
01
01
01   SEE IF THERE IS A DDSIOA
01
01   IF THERE IS AN IOA
01
01
02     SEE IF BACKUP DCB IS OPEN
02
02     IF BACKUP DCB IS OPEN
03       ASYNCHRONOUSLY CLOSE THE BACKUP DCB
03
03
02     ENDIF
01   ENDIF
01   UPCATE CDSTATUS RECORD
   ENDIF
```

Figure 3-122. DPPSUNLK

```
       ' UNLCCK A DDS '
       GET ADDRESS OF CTLA
       IF CALLED FROM ETXR, THEN
01       GET A(CTLA)
       ENDIF
       GET CURRENT LECB
       GET NEXT LECB
       STORE IN CTLA
       IF THERE IS ANOTHER LECB
01       POST THAT LECB
       ELSE
01       GET 1ST SECB, IF ANY
01       ZERO SHARE COUNTER
01       WHILE THERE ARE MORE SECBS
02         POST THAT SECB
02         INCREMENT SHARE COUNTER
02         GET NEXT SECB, IF ANY
01       ENDDC
01       SET SECB WORD (WITH SHARE COUNTER)
       ENDIF
       IF DDS HCLD IS OFF, THEN
01       SRTOS UNLOCK THAT DDS LOCK/SHARE CHAIN
       ENDIF
       IF CALLED FRCM ETXR,THEN
01       GET A(TCBC) DIRECTLY
       ELSE
01       CALL DPPSADDX TO GET A(TCBC)
       ENDIF
```

Figure 3-123. DPPSUNSH

```
   'UNSHARE A DDS'
   GET ACCRESS OF CTLA
   IF CALLED FROM ETXR, THEN
01   GET A(CTLA)
   ENDIF
   GET SHARE CCUNTER
   DECREMENT SHARE COUNTER BY 1
   IF NC MORE SECBS
   IF LOCK IS WAITING
01   SET LCCK FLAG ON
01   POST THE WAITING LOCK
   ENDIF
   IF DDS HOLD IS CFF, THEN
01   SRTOS UNLCCK THAT DDS LOCK/SHARE CHAIN
   ENDIF
   IF CALLED FROM ETXR,THEN
01   GET A(TCBC) DIRECTLY
   ELSE
01   CALL CPPSADCX TO GET A(TCBC)
   ENDIF
```

Figure 3-124. DPPSWRST

```
   GET DDSCTLHD
   IF MODE IS NOT READONLY, THEN
01   SET MAX BLOCKSIZE FOR DDSTATUS
01   OPEN DDSTATUS
01   IF OPEN WENT O. K. , THEN
02     ESTABLISH SYNAD ADDRESS
02     CALCULATE DDSTATUS RECORD LENGTH
02     WRITE THE CTLA'S
02     CHECK THE WRITE
02     CLOSE DDSTATUS
02     OUTPUT MESSAGE INDICATING
02     GOOD UPDATE
02     ZERO THE RETURN CODE
01   ELSE
02     OUTPUT DDSTATUS NOT OPEN MESSAGE
02     SET RETURN CODE TO 8
01   ENDIF
   ELSE
01   OUTPUT READONLY MESSAGE
01   SET RETURN CODE TO 4
   ENDIF
   RETURN TO THE USER
   ENTER SYNAD PROCESSING HERE
   CLOSE DDSTATUS
   OUTPUT SYNAD MESSAGE
   SET RETURN CODE TO 16
   RETURN TO THE USER
```

Figure 3-125. DPPSXTCB

```
    FUNCTION:  TO SEE IF THE INPUT(CURRENT) TCB HAS A DDSXTCBC, AND
    IF YES, RETURN ITS ADDRESS IN $1, OR IF NOT,
    ALLOCATE, INITIALIZE, AND CHAIN A NEW DDSXTCBC AND
    RETURN ITS ADDRESS IN $1. ALL DDSXTCBC'S ARE CHAINED
    OFF THE SCVT.
    INPUTS: PTR TO ( 0 OR INPUT TCB), IF 0,GET CURREN TCB FROM THE
    CVT-TCB-SCVT PATH.
    OUTPUTS: $1 WILL CONTAIN THE ADDRESS OF THE DDSXTCBC FOR THE
    INPUT TASK
    'LOCATE/ALLOCATE A DDSXTCBC FOR AN INPUT TASK'
    IF
01    THE TCBX IS NOT INPUT,THEN GET THAT ADDRESS VIA CVT-JSTCB
    ENDIF
    IF SRTCS JOB STEP,THEN
01    GET A(TCBX) INSTEAD OF TCB
    ENDIF
    STRTSRCH
01    WHILE THERE ARE MORE TCBC'S
01    GET NEXT TCBC
    BGNWHILE
    EXITIF FOUND DDSXTCBC FOR INPUT TASK
    ORELSE
    ENDLOOP
01    GET CORE FOR A NEW TCBC AND CHAIN IT TO THE OTHERS
    ENDSRCH
```

Figure 3-126. DPPTCBGT

```
    DPPTCBGT -  TYPE 1 SVC ROUTINE FOR CONTROL BLOCK GET                *
    IF THIS IS A CBGET REQUEST
01    CALC # OF BLKS NEEDED - LNTH/32
01    GET LAST USED PSCB
01    *     PSCB=PROTECTED STORAGE CTRL BLOCK
01    SET STOP IND FOR CIRCULAR CHAIN
01    STRTSRCH SEARCH PSCB CHAIN TILL STOPIND
01    EXITIF STOP WHEN PSCB WITH ENUF CORE FOUND
02      IF # FREE NE # AVAIL-A PSCB MUST BE
03        *           CREATED
03        CALC WHERE TO PUT NEW PSCB
03        PUT PSCBID IN NEW PSCB
03        ADD NEW PSCB TO CIRCULAR CHAIN
02      ENDIF
02      COMPLIMENT THE BLOCK COUNT
02      PUT COMPLIMENTED COUNT IN BLOCK
02      PUT GOTTEN CORE ADDR IN REG 1
02      TURN OFF STOP IND
02      UPDATE LAST USED PTR
01    ORELSE
02      GET ADDR OF NEXT PSCB
02      CHECK FOR STOP IND
01    ENDLOOP
02      SET NO CORE AVAIL RET CODE
01    ENDSRCH
01    IF CBGET CORE WAS AVAILABLE
02      CLEAR GOTTEN CORE TO ZEROS
02      RETURN
01    ENDIF
    ELSE
01    CHECK ADDR FOR VALIDITY
01    IF A PSCB - A VALID ADDRESS
02      *            WAS PASSED TO BE FREED
02      COMPLIMENT THE FREE COUNT
02      DEALLOCATE BY RESTORING FREE CNT
02      GET NEXT PSCB
02      IF IT HAS FREE BLOCKS
03        ADD THEM  TO CURRENT PSCB
03        REMOVE OTHER PSCB FROM CHAIN
02      ENDIF
02      GET PREVIOUS PSCB
02      IF IT HAS FREE BLOCKS
03        ADD CURRENT FREE TO PREV FREE BLKS
03        REMOVE THIS PSCB FROM PSCB CHAIN
02      ENDIF
02      RESET LAST USED PTR
01    ELSE
02      SET INVALID ADDR RETURN CODE
01    ENDIF
    ENDIF
    RETURN
```

Figure 3-127. DPPTCSVC

```
DPPTCSVC - TYPE 1 SVC ROUTINE FOR CHAIN                          *
IF
ENDIF
VALIDITY CHECK INPUT ADDRESSES : MUST BE WITHIN PARTITION(S) *
*                                CHECK BYPASSED IF PROGRAM  *
*                                  ISSUEING SVC HAS PK=C OR  *
*                                  SUPVR STATE               *
```

Figure 3-128 (1 of 3). DPPTDLMP

```
   *********************************************************************
   *                                                                   *
   *  MODULE NAME       - DPPTOLMP                                     *
   *                                                                   *
   *  DESCRIPTIVE NAME - TASK MANAGEMENT * CYNAMIC LOAD MODULE PURGE   *
   *                                                                   *
   *********************************************************************
   GET INPUT PARAMETERS
   IF NO TOME SPECIFIED        DR #5192
01   TAKE DEFAULT OF 2 SECS   DR #5192
   ELSE DR #5192
01   IF NO TIME WAS SPECIFIED
02     TAKE DEFAULT CF 2 SECCNDS
01   ENDIF
   ENDIF DR #5192
   IF TIME VALUE GT MAX
01   ISSUE MESSAGE DPP019
   ELSE - IF TIME VALUE VALID
01   IF NC MODULES WERE PASSED      DR #5192
02     ISSUE MESSAGE 22   DR #5192
01   ELSE DR #5192
02     UNTIL SEARCH ENTIRE LIST          DR #5192
03       IF VAIID MODULE NAME ADCR      DR #5192
04         IF THIS IS FIRST VALID ADDR    DR #5192
04         ENDIF DR #5192
03       ENDIF DR #5192
02     ENDCO R               DR #5192
02     IF NO VALID ADDRS                DR #5192
02     ELSE DR #5192
03       DEFINE LCCK
03       LOCK OTHER DLMP-REQUESTS OUT
03       ISSUE MESSACE DPP020
03       SET ECB COUNT TO ONE
03       ISSUE STIMER
03       GET INDEPENDENT TASK CHAIN ORIGIN
```

Figure 3-128 (2 of 3). DPPTDLMP

```
03        DO OPTDLMP1 - TCBX-LCB LSER SCAN
03        GET DEPENDENT TASK CHAIN ORIGIN
03        CO DPTDLMP1 - TCBX-LCB USER SCAN
03        GET TMCT-LCB CHAIN ORIGIN
03        WHILE MCRE LCB'S ARE CHAINED
04          UNTIL ALL MODULE NAMES EXAMINED
C5            IF NAME EQUAL TO LCBEPNAM
C6              SET FLAG MODULE PURGE REQUESTED
05            ENDIF
04          ENDDC UNTIL BXLE
03        BGNWHILE
04          GET ADDR OF NEXT LCB IN CHAIN
03        ENDDO WHILE LCB
03        TURN CFF STIMER FLAG
03        WHILE ECB COUNT DOES NOT GO TC ZERO
04          WAIT FOR  X  EVENTS OUT OF LIST
04          IF STIMER EXPIRED
05            SET STIMER FLAG
05            CAUSE EXIT OF BCT-LOOP
04          ENDIF STIMER
03        BGNWHILE
04          INCR NUMBER OF EVENTS TO WAIT FOR
03        ENDDO WHILE BCT
03        IF ALL POSTS RECEIVED IN TIME
04          CANCEL THE CUTSTANDING STIMER
03        ENDIF
03        GET INDEPENDENT TASK CHAIN ORIGIN
03        DO DPTDLMP2 - TCBX-LCB PURGE SCAN
03        GET DEPENDENT TASK CHAIN ORIGIN
03        DO DPTDLMP2 - TCBX-LCB PURGE SCAN
03        TURN CFF INTERNAL DELETE FLAG
03        GET TMCT-LCB CHAIN ORIGIN
C3        WHILE MORE LCB'S ARE CHAINED
04          IF STIMER HAS NOT EXPIRED
05            IF PURGE REQUESTEC
06              SET FLAG DELETE REQ BY SMCN IN LCB
06              SET INTERNAL DELETE FLAG
05            ENDIF
04          ELSE - IF STIMER HAS EXPIRED
05            TURN OFF PURGE FLAG IN THE LCB
04          ENDIF STIMER EXPIREC
03        BGNWHILE
04          GET ADDR OF NEXT LCB IN CHAIN
03        ENDDO
03        IF STIMER HAS NOT EXPIRED
04          IF INTERNAL DELETE FLAG SET
05            CLEAR THE ECB
05            SET FLAG DELETE REQ BY SMON IN TMCT
05            POST CPPTSMON FOR DELETE SERVICE
04          ENDIF
```

Figure 3-128 (3 of 3). DPPTDLMP

```
04          IF ANY ECB WAS SET UP
05            WAIT CN TFE ECB LIST
04          ENDIF
03        ENDIF STIMER EXPIRED
03        GET INDEPENDENT TASK CHAIN ORIGIN
03        DO DPTDLMP3 - TCBX POST SCAN
03        GET DEPENDENT TASK CHAIN ORIGIN
03        DO DPTDLMP3 - TCBX POST SCAN
03        IF STIMER EXPIRED
04          GET ADDR OF FIRST ECB
04          UNTIL ALL ECB'S EXAMINED
05            IF ECB WAS NOT POSTED
06              IF MODULE NAME PRESENT
07                ISSUE MESSAGE DPP021
07                CLEAR OUT MODULE NAME
06              ENDIF
05            ENDIF ECB NOT POSTED
04          ENDDO UNTIL BXLE
04          LOAD MSG # FOR  PURGE ABANDONED
03        ELSE - IF STIMER HAD NOT EXPIRED
04          LOAD MSG # FOR  PURGE CCMPLETE
03        ENDIF STIMER EXPIRED
03        ISSUE MESSAGE
03        UNLOCK - WE ARE THROUGH
03        RELEASE THE LCCK DEFINITION
02      ENDIF TIME VALID
01    ENDIF NO VAIIC MCDULE NAMES                                     DR #5192
    ENDIF INVALID PARM LIST - NC NAMES SUPPLIED                       DR #5192
    RETURN
    COPY DPTDLMP1 - TCBX-LCB USER SCAN
    COPY DPTDLMP2 - TCBX-LCB PURGE SCAN
    COPY CPTDLMP3 - TCBX POST SCAN
    COPY DPTDLMP4 - STIMER EXIT ROUTINE
    COPY DPTDLMP5 - ASYNC DELETE ROUTINE
```

Figure 3-129. DPPTDSVC

```
   ************************************************************************
   *                                                                      *
   *  MODULE NAME      - DPPTDSVC                                         *
   *                                                                      *
   *  DESCRIPTIVE NAME - TASK MANAGEMENT * DPATCH TYPE 1 SVC ROUTINE      *
   *                                                                      *
   ************************************************************************
   LOAD DPATCH TYPE CODE
   LOAD ADDR OF TCBXNAME FIELD
   COPY DPTDSVC1 - VALIDITY CHECK
   IF RETURN CODE LOW
01   IF DPI FLAG IS SET
02     SET RC FOR TASK BEING REMOVED
01   ELSE
02     IF THIS WAS DPATCH=I
03       SET FLAGS DPI + DPU IN TCBX
03       GET ABEND CODE
03       GET ADDR OF ABTERM FROM CVT
03       ABTERM THE DPATCHED TASK
02     ELSE
03       IF DPW FLAG IS SET
04         SET RC FOR ALREADY DPATCHED=W
03       ENDIF DPW
03       IF DPU FLAG IS SET
04         SET RC FOR ALREADY DPATCHED=U
03       ENDIF DPU
03       IF RETURN CODE LOW
04         IF THIS WAS DPATCH=U
05           SET DPU FLAG
04         ELSE
05           IF THIS WAS DPATCH=W
06             SET DPW FLAG
05           ELSE
06             IF THIS WAS DPATCH=C
07               IF TASK DORMANT FLAG SET
08                 SET DPC FLAG
07               ELSE
08                 SET RC TASK NOT DORMANT
07               ENDIF TASK DORMANT
06             ELSE
07               SET RC INVALID INPUT PARMS
06             ENDIF DPATCH C
05           ENDIF DPATCH W
04         ENDIF DPATCH U
04         IF RETURN CODE LOW AND
04         IF NO DPPTSMON REQ PENDING
05           IF DPPTPMON NOT POSTED
06             POST DPPTPMON
05           ENDIF POSTBIT
04         ENDIF RC LOW AND NO SMON REQ
03       ENDIF RC LOW
02     ENDIF DPATCH I
01   ENDIF DPI SET
   ENDIF RC ZERO
   RETURN
   COPY DPTPSVC2 - ADDRESS CHECK
```

Figure 3-130 (1 of 3). DPPTETXR

```
    ***********************************************************************
    *                                                                     *
    *   MOCULE NAME      - DPPTETXR                                       *
    *                                                                     *
    *   DESCRIPTIVE NAME - TASK MANAGEMENT * ENC-OF-TASK-EXIT ROUTINE     *
    *                                                                     *
    ***********************************************************************
    LOAD ADDR OF TCB WE'RE RUNNING UNDER
    IF IT'S NOT JOBSTEP = SMON'S TCB
01    ISSUE ABEND USER CODE 64
    ENDIF
    GET XCVT ADCR FROM DPPTSMON'S TCBX
    GET ACDR OF FIRST LCCK CONTROL BLOCK
    WHILE MORE LOCKCBLK'S ARE CHAINED
01    IF EXITING TASK HAS CCNTRCL CF THIS RESOURCE
02      CHANGE THE TCB ADDR TO SMON'S TCB
02      UNLCCK THE RESOURCE
01    ENDIF
01    GET ADDR CF NEXT LOCKCBLK ON CHAIN
    ENDDO
    IF THIS PARTITICN IS SLAVE PARTN
01    GET MASTER PARTN XCVT ADDR
01    GET MASTER PARTN SCVT ADDR
01    GET ADDR OF FIRST LOCKCBLK ON CHAIN
01    WHILE MORE LCCKCBLK'S ARE CHAINED
02      IF EXITING TASK HAS CNTRL CF THIS RESCURCE
03        CHANGE THE TCB ADDR TO SMON'S TCB
03        UNLCCK THE RESOURCE
02      ENDIF
02      GET ADDR OF NEXT LOCKCBLK CN CHAIN
01    ENDDO
    ENDIF
    IF TCBXDCVT-FIELC WITHIN PARTITION
01    GET XCVT ACCR FRCM EXITING TASK'S TCBX
01    IF XCVT ADDRESSES MATCH
02      IF CURRENT WQE PRESENT
03        IF CWQ ADCR OUTSIDE PARTITION OR
03        IF WQE DOES NOT POINT BACK TO TCBX
04          ISSUE MESSACE DPP010
04          CLEAR WQE ADDR
03        ELSE - CURRENT WQE ADCR VALID
04          LOAC ACCR CF LCB FRCM WQE
03        ENDIF
02      ENDIF CURRENT WCE
02      IF TCB CCMPLETICN CODE FIELC NZERO
03        * TASK TERMINATED ABNORMALLY
03        IF IT WAS MSG OUTPUT TASK
04          ISSUE WTO MSG DPP011
03        ENDIF MSG OUTPUT TASK
```

Figure 3-130 (2 of 3). DPPTETXR

```
03      IF CURRENT WQE PRESENT
04        MOVE EP-NAME INTO MSG TEXT
03      ELSE
04        CLEAR MSG TEXT AREA WITH BLANKS
03      ENDIF CURRENT WQE
03      ISSUE MESSAGE DPP012
02    ELSE - TCB COMPLETION CODE FIELD ZERO
03      IF EXIT NOT CAUSED BY DPPTPMON
04        ISSUE MESSAGE DPP018
03      ENDIF
02    ENDIF COMPL CODE
02    IF EXIT NOT CAUSED BY DPPTPMON
03      IF CURRENT WQE PRESENT
04        IF MODULE WAS LOADED BY ABENDING TASK
05          * IT WILL BE DELETED BY
05          * TASK TERMINATION OF CS/VS
05          SET LCB UNRESOLVED
04        ENDIF
04        GET COMPLETION CODE FROM TCB
04        SET ECB CC FOR ABNORMAL TERMINATION
04        STORE ECB COMPLETION CODE INTO WQE
04        IF WQE ABENDED FLAG ZERO
05          * THIS WQE DID NOT CAUSE ABEND BEFORE
05          SET FLAG WQE ABENDED
05          GET CLEANUP WORK QUEUE ORIGIN
05          WHILE MORE WQE'S ARE CHAINED
06            KEEP ADDR OF THIS WQE
05          BGNWHILE
06            GET ADDR OF NEXT WQE
05          ENDDO
05          CHAIN OUR WQE TO END OF CLEANUP-WQ
05          CLEAR CHAIN WORD IN OUR WQE
04        ELSE - WQE CAUSED ABEND BEFORE
05          EXEC WQE-DELETE SVC ROUTINE
04        ENDIF WQE ABENDED
03      ENDIF CURRENT WQE
03      RESET WQ ACTIVE FLAG IN TCBX
03      SET FLAGS TCB + CHAP REQUIRED
03      CLEAR SMON CHAIN WORD IN TCBX
03      GET TMCTSMON CHAIN ORIGIN
03      WHILE MORE TCBX'S ARE CHAINED
04        GET ADDR OF NEXT TCBX ON CHAIN
03      ENDDO
03      CHAIN TCBX TO DPPTSMON'S REQUEST CHAIN
03      CLEAR ADDR OF DEAD TCB FROM TCBX
03      CLEAR POSTBIT
03      POST DPPTSMON FOR SERVICE
02    ELSE - EXIT WAS CAUSED BY DPPTPMON
03      CBFREE THE TCBX
02    ENDIF PMON CAUSED EXIT
```

Figure 3-130 (3 of 3). DPPTETXR

```
02       CLEAR REG 15
01     ELSE - XCVT ACCR DON'T MATCH
02       * CANNOT TRUST TCBX
02       LCAD REG 15 NZERO
01     ENDIF XCVT ADDR MATCH
   ELSE - TCBXDCVT FIELD OUTSIDE PARTITION
01     LOAC REG 15 NZERO
   ENDIF
   IF REG 15 NZERO - BAD TCBX ADDRESS
01     ISSUE MESSAGE DPP013
   ENDIF
   DETACH CEAD TCB
   RETURN
```

Figure 3-131. DPPTGWFW

```
DPPTGWFW - BRANCH SUBROUTINE FOR GETWA/FREEWA                               *
ISSUE A GETWA SVC
IF RETURN CODE IS PLUS
RETURN TO CALLER
ELSE
IF FREEMAIN SERVICES REQUIRED
FREE THE EXTRA GETWA CORE
ZERO RETURN CODE
RETURN TO CALLER
ELSE
GET ACCRESS OF GFMB TO BE EXPANDED
GET LENGTH REQUIRED FOR GETMAIN
GET ADDITIONAL KEY 0 STORAGE FOR SIZE OF GETWA REQUIRED
GET CCNTROL BLCCK STCRAGE
IF CBGET CORE NOT AVAILABLE
FREE THE STORAGE GOTTEN FOR GETWA
FREE THE SAVE AREA
SET RETURN CODE=12
RETURN TO CALLER
ELSE
PUT GETWA LOW ADDRESS IN THE GFCB
PUT GETWA HI ADDRESS IN THE GFCB
POINT NEW GFCB TO ITS GFMB
POINT GFCB TO FIRST GFBE
PUT INIT # OF BLCCKS IN GFCB
PUT INIT FREE COUNT IN GFCB
UPDATE TOTAL # FREE BLKS IN GFMB
ADD NEW GFCB TO CHAIN OFF TMCT
ADD GFCB TO GFMB CHAIN
RETURN TO EP (DPPTGWFW) TO RETRY GETWA
ENDIF
ENCIF
ENDIF
ENDIF
```

Figure 3-132 (1 of 2). DPPTIMPS

```
   DPPTIMPS - CUMP/NO CUMP FACILITY IMP INTERFACE
   DEFINE A RESCURCE FCR LOCK
   LOCK RESOURCE
   IF NO SYSTEM DUMP CONTROL BLCCK,THEN
01   GETMAIN FOR DUMP CNTL BLOCK
01   ENTER SUPSTATE
01   SAVE A(DUMP CNTL BLOCK) IN SCVT
01   EXIT FROM SUPSTATE
   ENDIF
   IF PARAMS WERE PASSED,THEN
01   GET CPTION PARAM ADDRESS
01   IF OPTION PASSEC,THEN
02     MCVE IN REQUESTED OPTION
02     UNTIL ALL OPTIONS CHECKED,DO
03       BUMP TO NEXT VALID OPTION
02     ENDDC
02     IF NOT NCRMAL CPTION,THEN
03       SET ERRCR MESSAGE
02     ELSE
03       MCVE CPTICNS INTO DEFAULT
02     ENDIF
01   ENDIF
01   IF VALID CPT OR DEFAULT SEL,THEN
02     IF LOAD MODULES,THEN
03       CALC AND SAVE NUM LOAD MODULES
03       UNTIL ALL LOAD MODULES PROCESSED,DO
04         IF VALID LM ADDR,THEN
05           MOVE NAME INTO WORK AREA
05           IF INVALIC LM NAME,THEN
06             ISSUE ERROR MESSAGE
05           ELSE
06             SAVE NAME LENGTH
06             WHILE NOT END CF CHAIN,AND
06             WHILE NAME LOW,DO
07               INCR DUMP CNTL BLOCK PTR
06             ENDDO
06             IF NOT END OF CHAIN,AND
06             IF NAME FCUND,THEN
07               RESET FLGS
06             ELSE
07               GET NEW DUMP CNTL BLOCK
07               CHAIN IN NEW CUMP CNTL BLCCK
06             ENDIF
05           ENDIF
04         ENDIF
03       ENDDO
02     ENDIF
01   ENDIF
   ENDIF
```

Figure 3-132 (2 of 2). DPPTIMPS

```
   IF VALID OPTION,THEN
01   IF DEFAULT CHANGED,THEN
02     RESET DEFAULT FLGS
02     ZERO TFE CFAIN PCINTER FROM SYSBLOCK
02     WHILE NOT END OF CHAIN,DO
03       FREEMAIN DUMP CNTL BLCCK
02     ENDDO
02     ISSUE MESSAGE
01   ELSE
02     ISSUE MESSAGE
01   ENDIF
   ELSE
01   ISSUE MESSAGE
   ENDIF
   UNLOCK RESOURCE
   RETURN
```

Figure 3-133 (1 of 2). DPPTPMON

```
    **********************************************************************
    *                                                                    *
    *  MCDULE NAME       - DPPTPMON                                      *
    *                                                                    *
    *  DESCRIPTIVE NAME - TASK MANAGEMENT * PATCH MCNITOR ROUTINE        *
    *                                                                    *
    **********************************************************************
    WAIT FOR THE TCBX ADDR TO BE FILLED IN BY
    * THE MCTHER TASK  (DPPINIT OR DPPTSMCN)
    IF NO RESOURCE TABLE PRESENT
01    GET VIRT STORAGE FOR RESTBL AND WCRKAREA
01    CLEAR WCRKAREA PLUS RESOURCE TABLE
01    STORE RESCURCE TAELE ADDR INTO TCBX
    ELSE
01    GET ADDR CF WORK AREA
    ENDIF NC RESOURCE TABLE
    IF STAE TO BE ISSUED            PRF#166
    ENDIF PRF#166
    UNTIL TASK TERMINATN REQUIREC
01    IF THIS TASK ATTACHED BY DPPINIT
02      WAIT FOR FIRST PATCH
01    ENDIF
01    MAKE IT LCCK LIKE ATTACFED BY DPPINIT
01    UNTIL ANY DPATCH DO
02      WHILE WCE'S CHAINED TO TCBXWQ AND
02      WHILE NO DPATCH=U
03        TOP WQE BECCMES THE CURRENT
03        CECHAIN TOP WCE
03        IF CURRENT QUEUE LENGTH NCT ZERO
04          DECREMENT CURRENT QUEUE LENGTH
03        ENDIF CCL NZERO
03        DO DPTPMON3 - LOAD USER'S ROUTINE
03        DO DPTPMON4 - EXEC LSER'S PROGRAM
03        DO DPTFMCN5 - PERFOPM WQE-CLEANUP
02      BGNWHILE
03        WHILE WCE'S CHAINED TC TCBXCUWQ
04          TOP WQE BECCMES CURRENT
04          DECHAIN TOP WQE
04          IF DDS WAS INVOKED
05            EXEC DDS CLEANUP ROUTINE
04          ENDIF
04          DO DPTPMCN5 - PERFOFM WQE CLEANUP
03        ENDDO WHILE TCBXCUWQ NZERO
02      IF QP TCBX THEN
03        DO DPTP SELECT WQ FROM QH
02          ENDDO
02      ENDDO WHILE TCBX WQ AND NO DPATCH U
```

Figure 3-133 (2 of 2). DPPTPMON

```
02      IF NO DPATCH CF ANY KIND
03        CLEAR CWQ POINTER
03        CLEAR POSTBIT BEFORE WAIT
03        SET FLAG TASK DCRMANT
03        WAIT FOR NEXT PATCH, CPATCH OR REPATCH
03        TURN OFF TASK DORMANT FLAG
02      ENCIF
01    ENDDO UNTIL ANY DPATCH
01    SET FLAG TASK IS BEING REMOVED
01    IF DPATCH WORK QUEUE IS NCT EMPTY
02      DPATCH WQE BECCMES THE CURRENT
02      CLEAR DWC POINTER
02      DO DPTPMON3 - LCAD USER'S ROUTINE
02      DO DPTPMON4 - EXEC USER'S PROGRAM
02      DO DPTPMCN5 - PERFORM WQE-CLEANUP
01    ENDIF
01    CLEAR CWQ PCINTER
01    COPY DPTPMON1 - TCBX-CLEANUP AFTER CPATCH
   ENDDO UNTIL TASK TERMINATION REQ
   FREEMAIN WORK AREA AND RESOURCE TBL
   CLEAR RESOURCE TABLE ADDR FROM TCBX
   ISSUE SVC EXIT
   COPY DPTPMON2 - HI LEVEL LANG INTERFACE
   COPY DPTPMON3 - USER ROUTINE LOAD
   COPY DPTPMON4 - USER ROUTINE EXEC
   COPY DPTPMON5 - WQE-CLEANUP
   COPY DPTPMON6 - QP/QH INTERFACE
```

Figure 3-134 (1 of 2). DPPTPSVC

```
    ***********************************************************************
    *                                                                     *
    *  MOCULE NAME       - DPPTPSVC                                       *
    *                                                                     *
    *  DESCRIPTIVE NAME - TASK MANAGEMENT * PATCH TYPE 1 SVC ROUTINE      *
    *                                                                     *
    ***********************************************************************
    LOAD ADDR OF PROBLEM PARM LIST
    LCAD ACCR OF SUPERVISOR PARM LIST
    COPY DPTPSVC1 - VALIDITY CHECK
    IF INPUTS ARE VALIC
01    IF TCBX WAS SPECIFIED
02      IF GIVEN NAME SAME AS IN TCBX
03        DO DPTPSVC3 - BUILD WQE AND LCB
02      ELSE
03        SET RC FOR INVALID TCBX ADDRESS
03        ZERO TCBX= ADCR TO CAUSE CHAIN SRCH
02      ENDIF GIVEN NAME SAME
01    ENDIF TCBX SPECIFIED
01    IF NC TCBX ADDRESS SPECIFIED
02      IF TCBXNAME SPECIFIED (INDEP TASK)
03        UNTIL FIND-LCCP COUNTER ZERO
04          DECREMENT LCOP COUNT
04          RESET TCBXNAME NCT FCUND FLAG
04          GET ORIGIN CF INDEP TASK CHAIN
04          STRTSRCH WHILE MORE TCBX'S CHND
04          EXITIF TCBXNAME FCUND
05            DO DPTPSVC3 - BUILD WQE AND LCB
05            TURN OFF FIND FLAG
05            SET LOOP CTR ZERC - CAUSE EXIT
04          ORELSE
05            GET NEXT TCBX IN CHAIN
04          ENDLCCP - NCT FOUND
C5            SET TCBXNAME NOT FCUND FLAG
04          ENDSRCH
04          IF FIND WAS SPECIFIED
05            SEE IF OTHER PTN IS STILL ACTIVE
05            IF
06              TURN CFF FIND FLAG
05            ELSE
06              SWITCH PARTITICNS
C5            ENDIF
04          ENDIF
03        ENDDO UNTIL FIND-LOOP
03        IF TCBXNAME NCT FOUND FLAG SET
04          DO DPTPSVC4 - BUILD TCBX
04          IF RETURN CCDE LOW
05            CHAIN TO INDEP TASK CHAIN
04          ENDIF
```

Figure 3-134 (2 of 2). DPPTPSVC

```
03       ENDIF NCT FCUND ON CHAIN
02     ELSE - NO TCBXNAME SPEC (DEPENDENT)
03       DO DPTPSVC4 - BUILD TCBX
03       IF RETURN CCDE LOW
04         SET DPATCH W FLAG TO CAUSE
04         *  DPPTPMON TO DELETE THE TASK AFTER ONE EXEC
04         CHAIN TO DEPENDANT TASK CHAIN
03       ENDIF
02     ENDIF TCBXNAME SPEC
01   ENDIF TCBX NOT SPEC
   ENDIF INPUTS VALID
   IF INDEPENDENT TASK AND
   IF RETURN CODE LE 8
01   TELL USER THE TCBX-ADDRESS IN REG 1
   ELSE
01   CLEAR REG 1
   ENDIF
   RETURN
   COPY DPTPSVC2 - ADDRESS CHECK
   COPY DPTPSVC3 - BUILD WQE AND LCB
   COPY DPTPSVC4 - BUILD TCBX
```

Figure 3-135. DPPTPWQE

```
    BGNSEG  DPPTPWQE  - PURGE WORK QUEUE ROUTINE
01    VERIFY INPUT PARAMETER ADDRESSES
01    IF ANY ARE INVALID
02      SET RETURN CODE
01    ELSE
02      IF TASKNAME IS ZERO
03        USE CURRENT TASK'S TCBX
02      ELSE
03        SEARCH FOR TCBX OF SPECIFIED TASK NAME
02      ENDIF
02      IF DPATCH IN PROGRESS FOR TASK
03        SET RETURN CODE
02      ELSE
03        IF TASK IS DORMINANT
04          SET RETURN CODE
03        ELSE
04          IF CURRENT WQ IS ONE OF THE WQ'S TO BE DELETED
05            FLAG TO BE PURGED
04          ENDIF
04          IF DPATCM WQ IS ONE OF THE WQ'S TO BE DELETED
05            REMOVE WQ
05            CHAIN ON CLEANUP WQ
04          ENDIF
04          UNTIL ALL WQ'S ON WQ CHAIN HAVE BEEN PROCESSED
05            IF  WQ IS ONE OF THE WQ'S TO BE DELETED
06              REMOVE WQ
06              CHAIN ON CLEANUP WQ
05            ENDIF
04          ENDDO
04          IF FREE SPECIFIED ON PURGEWQ
05            SET LENGTH AND ADDRESS IN LAST WQ TO BE PURGED
04          ENDIF
03        ENDIF
02      ENDIF
01    ENDIF
     ENDSEG  DPPTPWQE
```

FIGURE 3-135.1. DPPTQIMP

```
DPPTQIMP PROGRAM TC PRCCESS CS CCMMANCS ENTEREC THROUGH IMP
GETMAIN WCRK SPACE
IF PATCH IC IS 04
IF PROBL LEN IS 12 OR 16
MOVE USER SUPPLIEC PARAMETERS TC WORK SPACE
SET MASK BITS TC SELECT TCBX REQUESTEC BY FIRST PARAMETER
SET MASK BITS TO CHANGE FLAGS AS REQUESTED BY 2ND PARAMETER
IF THIRC PARAMETER ENTEREC AS 'PURGE' CR OMITTED
IF NO ERRCRS DETECTEC
CISABLE SYSTEM TC PREVENT BRCKEN CHAINS
WHILE TCBXNEXT,IS,NZERC RUN ENTIRE TCBX CHAIN
IF TCBX OF TYPE SELECTEC
CHANGE FLAG BITS ACCCRCING TC MASKS
SAVE TCBX ADDR IN WCRK SPACE
ENDIF
ENDCC
ENABLE SYSTEM
IF SELECTEC TCBX FOUND
UNTIL ALL SAVEC TCBX ACCR'S PROCESSEC
IF PURGE WAS ENTEREC
CISABLE SYSTEM
PURGE WORK STACKEC THIS TCBX
ENABLE
ENDIF
IF CPTICN WAS NCNSEC CR REL
POST TASKS THAT CAN PRCCESS WCRK
ENDIF
FCRMAT FLAG BITS FCR MESSAGE
OUTPUT MESSAGE 862 TC REPCRT STATUS
IF PARAMETER 2 WAS XREF
OUTPUT MESSAGE 863 TC REPCRT CCNNECTICNS
ENDIF
ENDDC
ELSE SELECTEC TCBX NCT FOUND
OUTPUT MESSAGE 864
ENCIF
ELSE ERRORS IN INPUT
OUTPUT MESSAGE 864 IDENTIFYING PARAMETER IN ERROR
ENCIF
ELSE 3RC PARAMETER IN ERROR
OUTPUT MESSAGE 864 IDENTIFYING PARAMETER IN ERRCR
ENCIF
ELSE PRCBL LENGTH BAC
OUTPUT MESSAGE 864 STATING PROBL LEN IS BAD
ENDIF
ELSE PATCH IC BAC
OUTPUT MESSAGE 864 STATING PATCH IC IS BAD
ENCIF
FREE WORK SPACE ANC EXIT
```

Figure 3-136. DPPTRGWA

```
DPPTRGWA - GETWA CCNTROL BLCCK TRANSFER ROUTINE                     *
CLEAR RETURN CODE REGISTER
STRTSRCH FIND GFCB OWNING CORE
EXITIF ADDR GE LOW FOR THIS GFCB
EXITIF AND LT HI
  VALIDITY CHECK INPUT ADDRESS - ENSURE IT IS ON A BLOCK BOUNDARY
  IF ADDRESS IS CN BLOCK BOUNDARY
    GET GFBE ADDR
    IF THIS BLOCK IS ALLOCATED
      DECHAIN THE GFBE FROM IT'S CURRENT
      IF BLOCK IS TO BE CHAINEC ON PC CHAIN
      ELSE
        DEFAULT TO AP CHAIN ORIGIN
        IF AT REQUEST
          GET AT CHAIN CRIGIN
        ENDIF
      ENDIF
      ADD GFBE TO NEW CHAIN
    ELSE
      SET INVALIC ADDRESS RET CODE
    ENDIF
  ELSE
    LOAD INVALID GETWA ADDRESS RET CODE
  ENDIF
ENDLOOP
  LOAD INVALID GETWA ADDRESS RET CODE
ENDSRCH
RETURN TO CALLER
```

Figure 3-137. DPPTRSVC

```
    *********************************************************************
    *                                                                   *
    *   MODULEE NAME      - DPPTRSVC                                    *
    *                                                                   *
    *   DESCRIPTIVE NAME - TASK MANAGEMENT * REPATCH TYPE 2 SVC ROUTINE *
    *                                                                   *
    *********************************************************************
    LOAD REPATCH TYPE CODE
    LOAD ADDR OF REPATCH PARM LIST
    CLEAR RETURN CODE REGISTER
    USE DPPTSMON'S TCBX
    IF REPATCH TYPE CODE IS NOT 0 NOR 1
01    SET RC FOR INVALID TYPE
    ELSE
01    IF REPL ADDR INVALID
02      SET RC FOR INVALID REPL ADDR
01    ELSE
02      CHECK ADDRESS OF REPL
02      IF NOT WITHIN EITHER PARTITION
03        SET RC FOR INVALID REPL
02      ELSE
03        IF PRTY REF NOT ' REPATCH'
04          SET RC FOR INVALID REPL
03        ELSE
04          IF REPATCH TYPE=EXEC SPECIFIED
05            LOAD PROBL ADDR FOR PATCH
05            LOAD SUPL  ADDR FOR PATCH
05            EXECUTE PATCH SVC RTN
04          ENDIF TYPE EXEC
04          IF REPATCH TYPE=PURGE SPECIFIED
05            IF FREE SPEC ON ORIGINAL PATCH
05            IF FLAG NOT SET BY PTIME
06              FREEMAIN USER'S AREA
05            ENDIF
04          ENDIF TYPE PURGE
04          GET ADDR OF REPL
04          GET REPL CHAIN ORIGIN
04          STRTSRCH WHILE MORE REPL'S CHAINED
04          EXITIF THIS IS THE ONE WE SEARCH FOR
05            DECHAIN REPL FROM CHAIN
04          ORELSE
04          ENDLOOP - NOT FOUND ON CHAIN
04          ENDSRCH
04          CB-FREE THE REPL
03        ENDIF
02      ENDIF
01    ENDIF
    ENDIF TYPE VALID
    RETURN
    COPY DPTPSVC2 - ADDRESS CHECK
```

Figure 3-138 (1 of 2). DPPTSMON

```
   ***********************************************************************
   *                                                                     *
   *  MODULE NAME       - DPPTSMON                                       *
   *                                                                     *
   *  DESCRIPTIVE NAME - TASK MANAGEMENT * SYSTEM MONITOR ROUTINE        *
   *                                                                     *
   ***********************************************************************
   WHILE NEVER ENDING LOOP
01   WHILE MORE TCBX'S ON REQUEST CHAIN
02     DECHAIN FIRST TCBX
02     COPY DPTSMON1 - LOAD COMMON MODULES
02     IF A NEW TCBX REQUIRES A TCB
03       IF CHAP REQUEST FLAG SET
04         GET PRTY FROM TCBX
03       ELSE
04         USE ZERO PRTY
03       ENDIF
03       CALCULATE PRIORITY FOR ATTACH
03       ATTACH A NEW DPPTPMON
03       STORE ADDR OF NEW TCB INTO TCBX
03       IF CHAP REQUEST FLAG IS ZERO
04         SET SAME PCSTCODE AS DPPINIT
03       ELSE
04         ZERO HI ORDER BYTE
03       ENDIF
03       CLEAR REQUEST FLAGS
03       POST TCBX ADDR INTO NEW TCB
02     ENDIF TCB REQ
02     IF CHAP REQUEST FLAG SET
03       CALCULATE PRIORITY FOR CHAP
03       CHAP DPPTPMON UP TO PROPER PRTY
03       CLEAR CHAP REQUEST FLAG
03       POST DPPTPMON
02     ENDIF CHAP REQ
01   ENDDO WHILE TMCTSMON NZERO
01   IF DELETE PROCESSING REQUIRED
02     GET TMCT-LCB CHAIN ORIGIN
02     WHILE MORE LCB'S ON TMCT-LCB CHAIN
03       IF DELETE REQ FLAG SET
04         IF SOMEONE IS USING THIS MODULE DR#5192
04         IF AND IT IT NOT DUMP       DR#5192
04         ELSE DR#5036
05           DECHAIN THE LCB
05           IF LOAD MODULE PURGE REQUESTED
06             SET PURGE FLAG IN TMCT
05           ENDIF
```

Figure 3-138 (2 of 2). DPPTSMON

```
05           CLEAR EP ADDRESS
C5           DELETE THE MODULE
05           CB-FREE THE LCB
05           START AT TOP OF CHAIN AGAIN
04         ENDIF DR#5036
03       ENDIF DEL REQ FLAG
02     BGNWHILE
03       GET NEXT LCB IN TMCT-LCB CHAIN
02     ENDDO WHILE TMCTLCBA NZERO
02     CLEAR DELETE RECUEST FLAG
02     IF PURGE FLAG SET IN TMCT
03       TURN OFF PURGE FLAG
03       PCST CPPTDLMP
02     ENDIF
01   ENDIF DEL PROCESSING
   BGNWHILE
01   WAIT FOR POST BY
01   * DPPTPSVC, DPPTWQDL OR DPPTPMON
01   CLEAR POSTBIT FRCM ECB
   ENDDO WHILE ENDLESS LOOP
```

Figure 3-139 (1 of 2). DPPTSTAE

```
     GET A(CVT)
     GET A(TCB POINTER)
     GET A(TCB)
     IF NOT STEP ABENC,TFEN
01     IF NOT USER ABEND,THEN
02       GET A(TCBX)
02       IF TCBX EXISTS,THEN
03         GET A(XCVT)
03         GET SAVE/WORK AREA
03         CLEAR WCRK AREA
03         GET A(SCVT)
03         DEFINE LOCK RESOURCE
03         LCCK RESOURCE
03         DO FINCRB-FIND RB FOR ABENDING MODULE
03         CO FINDLCB - FIND MODULE NAME IN LCB
03         DO FINDMOD - FIND MCDULE CN DMP CHAIN
03         CO STAEBLCK - PROCESS STAE BLOCK
03         UNLCCK
03         RELEASE RESOURCE
02       ELSE
03         SET NO TCBX FLAG
03         TAKE ERROR EXIT ALL
02       ENDIF
01     ENDIF
     ENDIF
     RETURN
     BGNSEG FINDRB - FIND RB FOR ABENDING MODULE
01     SKIP PAST PRB FOR STAE
01     UNTIL END CF RB CHAIN,OR
01     UNTIL END RB PROCESSING,DO
02       IF IT IS A PRB,THEN
03         IF NOT DPPTPMCN,THN
04           GET A(ENTRY POINT NAME
04           SAVE A(RB) IN WORK AREA
04           TURN CN END RB PROCESSING FLAG
03         ENDIF
02       ENDIF
02       GET A(NEXT RB)
01     ENDDO
     ENDSEG FINDRB    END OF SEGMENT TO SCAN RB CHAIN FOR NAME
     BGNSEG FINDLCB - FIND MODULE NAME IN LCB
01     GET A(TCBX)
01     GET A(WQE)
01     IF WORK QUEUE EXISTS,THEN
02       GET A(LCB)
02       IF LCB EXISTS,THEN
03         GET A(ENTRY POINT NAME
02       ELSE
03         SET NO LCB FLAG
03         TAKE ERROR EXIT  -  ALL
02       ENDIF
```

Figure 3-139 (2 of 2). DPPTSTAE

```
01      ELSE
02        SET NO WQE FLAG
02        TAKE ERROR EXIT  -  ALL
01      ENDIF
      ENDSEG FINCLCB -END OF SEGMENT TO FIND EP NAME IN LCB
      BGNSEG
01      IF CHAIN EXISTS,THEN
02        GET A(NAME IN PRB)
02        IF NOT PRB NAME,THEN
03          GET A(NAME IN LCB)
02        ENDIF
02        UNTIL MCDULE FCUND,OR
02        UNTIL END OF CHAIN,DO
03          GET NAME LENGTH
03          IF MODULE NAME FOUND,THEN
04            SET MODULE FOUND BIT
03          ENDIF
02        ENDDO
02        IF MODULE NCT FCUND,THEN
03          GET A(SYSTEM STAE BLOCK)
02        ENDIF
01      ELSE
02        SET NO SCVTCCHN FCUND BIT
02        TAKE ERRCR EXIT ALL
01      ENDIF
      ENDSEG FINDMOD - END OF ROUTINE TO FIND MODULE STAE BLOCK
      BGNSEG STAEBLCK - PROCESS STAE BLOCK
01      IF NCT MODULE STAE BLK,AND
01      IF ONEDUMP REQUESTED,THEN
02        DO BLDSTABK - BUILD STAE BLCCK
01      ENDIF
01      IF ONE DUMP REQUESTED,THEN
02        IF MAX DUMPS LE NUM DUMPS,THEN
03          INDICATE NODUMP
02        ENDIF
01      ENDIF
01      ADD 1 TC NUM DUMPS
01      ENTER SUPSTATE
01      PUT STAE ABEND FLGS IN TCBCMPF
01      EXIT FROM SUPSTATE
      ENDSEG STAEBLOK  - END STAE BLOCK PROCESSING
      BGNSEG BLDSTABK - BUILD STAE BLOCK
01      GETMAIN FOR NEW DMP CNTL BLOCK
01      FILL IN NEW STAE BLOCK
01      GET A(START OF CHAIN)
01      UNTIL END OF CHAIN,OR
01      UNTIL CLD NAME BIGGER,DO
02        SAVE A(PREVIOUS STAE BLCCK)
01      ENDDO
01      PUT NEXT POINTER IN NEW STAE BLOCK
01      PUT A(NEW BLOCK) IN PREV STAE BLOCK
      ENDSEG BLDSTABK - END SEGMENT TO BUILD NEW STAE BLOCK
```

Figure 3-140 (1 of 2). DPPTWQDL

```
   ***********************************************************************
   *                                                                     *
   *  MODULE NAME       - DPPTWCDL                                       *
   *                                                                     *
   *  DESCRIPTIVE NAME - TASK MANAGEMENT * WORK-QUEUE-ELEMENT-DELETE     *
   *                     TYPE 2 SVC ROUTINE                              *
   *                                                                     *
   ***********************************************************************
   LCAC ADDR OF WORK-QUEUE-ELEMENT
   IF ECB ADDR WAS SPEC WITH PATCH AND
   IF THIS WQE FELL CUT OF QUEUE ANC
   IF REPATCH OPTICN WAS SPECIFIED
01   CB-GET FOR REPATCH LIST REPL
01   IF CB-GET PASSED ZERO RETURN CODE
02     CHAIN REPL TO TMCTREPL CHAIN
02     STORE IT'S ADDR INTO ITSELF
02     MOVE XCVT ACDR
02     MOVE TCBXNAME
02     MCVE EP NAME
02     SET PRTY REF NAME TO ' REPATCH'
02     MOVE PATCH FLAGS
02     MOVE QUEUE LENGTH
02     MCVE PRTY VALUE (ABSOLUTE HERE)
02     MOVE ECB ADDRESS
02     MOVE FREE LENGTH
02     MOVE FREE ACCRESS
02     IF PROBL WAS MOVED INTO WQE
03       MCVE PROBL INTO REPL
03       GET ADCR OF PROBL IN REPL
02     ELSE
03       GET SUPPLIEC ADDR CF PROBL
02     ENDIF
02     STORE ACDR CF PROBL
02     LOAD REPL ACDR - FOR POST INTO ECB
02     INSERT ECB CC FOR REPL BUILT
01   ELSE - NCNZERO RETURN CODE FRCM CB-GET
02     INSERT ECB CC FCR NO CB-GET STORAGE
01   ENDIF REPL GET
   ELSE - NOT REPATCH CASE
01   IF FREE SPEC WITH PATCH AND
01   IF FLAG NOT SET BY PTIME
02     IF GETWA AREA
02     ELSE
03       FREEMAIN USER'S AREA
02     ENDIF
01   ENDIF
   ENDIF REPATCH CASE
   IF ECB ADDR NZERC
```

Figure 3-140 (2 of 2). DPPTWQDL

```
01     POST LSERS ECB
     ENDIF
     DECREMENT REQUEST CCUNT IN TCBX-LCB
     IF EP-DELETE WAS SPECIFIED
01     IF AN ASSOC LCB IS CN TMCT CHAIN
02       DECREMENT USE CCUNT IN TMCT-LCB
02       IF USE CCUNT IN TMCT-LCB IS ZERO
03         INDICATE THIS LCB TC BE DELETED
03         REQUEST DELETE PROCESSING BY SMON
03         IF DPPTSMCN NOT POSTEC
04           POST DPPTSMCN
03         ENDIF POSTBIT
02       ENDIF USECNT
01     ENDIF TMCTLCB
01     IF REQUEST CCUNT IN TCBX-LCB IS ZERC
02       GET LCB ADDR TC SEARCH FOR
02       GET TCBX-LCB CRIGIN
02       STRTSRCH WHILE MORE LCB'S CHAINED
02       EXITIF THIS IS THE LCB WE LCCK FOR
03         DECHAIN LCB FROM TMCT-LCB CHAIN
03         CB-FREE THE LCB
02       CRELSE
03         KEEP ADDR OF THIS LCB
03         GET NEXT LCB IN TMCT-LCB CHAIN
02       ENDLOCP
02       ENDSRCH
01     ELSE - REQUEST CCUNT IS NCT ZERO
02       GET TCBX-LCB ADCR
02       SET LCB UNRESOLVED
01     ENDIF REQEST COUNT ZERC
     ENDIF DELETE SPEC
     CB-FREE THE WQ-ELEMENT
     RETURN
```

Figure 3-141 (1 of 2). DPPTWSVC

```
     DPPTWSVC - TYPE 1 SVC ROUTINE FOR GETWA/FREEWA                          *
     GET THE POINTER TO THE TCBX
     IF NO TCBX
01     GET JOB STEP TCB ADDRESS
01     IF NON-SRTOS TASK AND JOB
02       PUT INVALID ADDRESS IN REG 1
02       RETURN TO USER
01     ENDIF
01     IF THIS IS A GETWA
02       CHANGE TI TYPE=PC REQUEST
01     ENDIF
     ENDIF
     GET THE POINTER TO THE XCVT
     GET THE POINTER TO THE CVT
     GET POINTER TO THE TASK MANAGEMENT CONTROL TABLE
     IF THIS IS A GETWA REQUEST
01     IF THE REQUEST IS NONZERO AND NOT NEGATIVE
02       STRTSRCH LOOP THRU TABLE LOOKING FOR SIZE
02       EXITIF REQ SIZ LT SIZ FOR GFMB CHAIN
03         COPY BLOCK-GET ROUTINE (DPTWSVC1)
02       ORELSE
03         UPDATE TO NEXT GFMB ENTRY
02       ENDLOOP
03         SET INVALID SIZE RETURN CODE
03         PUT INVALID ADDR IN REG 1
02       ENDSRCH
01     ELSE
02       SET RETURN CODE = 4
02       PUT INVALID ADDRESS IN REG 1
01     ENDIF
     ELSE
01     FREEWA  REQUEST  $1=ADR TO FREE
01     STRTSRCH FIND THE GFCB OWNS
02       *    THIS CORE
01     EXITIF ADDR TO BE FREED GE LOW FOR THIS
01     EXITIF *   GFCB AND LT HI
02       ENSURE THAT THE ADDRESS PASSED FALLS ON A BLOCK BOUNDARY FOR
02       *  THIS GFCB
02       IF ADDR TO BE FREED IS ON A BLOCK
03         *   BOUNDARY FIND ITS GFBE AND
03         *   DECHAIN IT
03         IF THE GFCB IS ALLOCATED - DEALLOCATE
04           *     IT AND INCREMENT FREE COUNT
04           IF NOT INITIAL ALLOCATION OF GETWA
05             IF ALL BLOCKS IN GFCB ARE FREE
06               TURN ON ALL FREE FLAG
06               IF GFCB NOT LAST ON CHAIN
```

Figure 3-141 (2 of 2). DPPTWSVC

```
07                  MOVE GFCB TO END OF CHAIN
07                  UNTIL
08                    IF
09                      UNTIL FIND END OF CHAIN
09                      ENDDO
08                    ELSE
08                    ENDIF
07                  ENDDO
06                ENDIF
05              ENDIF
04            ENDIF
04            IF FREE CNT GT INITIAL ALLOCATION
05              IF NOT INITIAL ALLOCATED STORAGE
06                IF ALL BLOCKS ARE FREE
07                  IF GFCB CAN BE FREED
08                    REMOVE GFCB FROM GFMB AND TMCT CHAINS
08                    FREE THE CONTROL BLOCK STORAGE
07                  ENDIF
06                ENDIF
05              ENDIF
04            ENDIF
03          ELSE
04            SET RETURN CODE = 4
03          ENDIF
02        ELSE
03          BLK NOT ON BLK BNDRY OR ATTEMPT TO FREE UNALLOCATED BLK
03          SET RETURN CODE = 4
02        ENDIF
01      ORELSE
01      ENDLOOP
02        SET RETURN CODE = 4
01      ENDSRCH
      ENDIF
      RETURN
      COPY BRANCH ENTRY CODE - DPTWSVC3
```

Figure 3-142. DPPUMSG

```
    DPPUMSG  MAIN SEGMENT * SYSTEM MESSAGES FINAL PHASE PROCESSOR *
01     GET INPUT DEFMSG PARAMETERS
01     IF DELETE OPTICN THEN
02       COPY DPPUMSG1
01     ELSE * IF ADC TEST OR REPLACE OPTION *
02       COPY DPPUMSG2
01     ENDIF
    ENDSEGMENT  DPPUMSG
```

Figure 3-143. DPPUMSG1

```
    DPPUMSG1 INCLUDED SEGMENT * MESSAGE FINAL PHASE DELETE ROUTINE *
01     DO UNTIL ALL SPECIFIED MESSAGES ARE DELETED
02       PRINT MESSAGE TC BE DELETED
02       EXECUTE MACRO TO DELETE MESSAGE
01     ENDDO
    ENDSEGMENT  DPPUMSG1
```

Figure 3-144. DPPUMSG2

```
    DPPUMSG2 INCLUDED SEGMENT * MESSAGE FINAL PHASE ADD TEST REPLACE
01                                        ROUTINE *
       DO UNTIL ALL SPECIFIED MESSAGES ARE ADDED TESTED OR REPLACED
01       FIND MESSAGE IN MESSAGE DATA SET
01       IF ADD CPTICN AND MESSAGE FCUND THEN
02         PRINT ERROR MESSAGE
01       ELSE
02         PRINT INPUTTED MESSAGE
02         IF MESSAGE FOUND IN MESSAGE CATA SET THEN
03           PRINT OLD MESSAGE
02         ENCIF
02         IF OPTION IS ADD OR REPLACE THEN
03           EXECUTE WRITE MACRC TO ADD MESSACE TO MESSAGE DATA SET
03           EXECUTE STOW MACRO TO ADD MSG NUMBER TO MSG D.S. DIRECTORY
02         ENCIF
01       ENDIF
       ENDDC
    ENDSEGMENT DPPUMSG2
```

Figure 3-145 (1 of 7). DPPXDBAS

```
    DPPXCBAS - CATA BASE FINAL PHASE PROCESSOR
    SAVE REGS
    GET SAVE/WORK AREA
    CLEAR BUFFER
    LOAD DPPXCBCA
    LCAD DPPXCBLG
    PRINT CATA BASE FPP ENTERED MSG
    OPEN PDS CCB(OUTPUT)
    IF OPEN,THEN
01    IF NC BLKSIZ,THEN
02      GET MAX BLKSIZE FOR DEVIDE
01    ELSE
01    ENDIF
01    GETMAIN INPUT/CUTPUT BUFFERS
01    CLEAR GETMAIN AREA
01    IF ARRAYS GENERATED,THEN
02      OPEN PDS READ DCB(INPUT)
02      DO GETMNDIR - GET NEW DIRECTCRY BUFFER
02      IF ADD CPTION,THEN
03        CO ACCARRAY - ADD NEW ARRAYS
02      ELSE
03        IF DELETE OPTION,THEN
04          TURN ON DELETE MODE BIT
04          DC DELARRAY - DELETE ARRAYS
03        ELSE
04          IF REPLACE CPTICN,THEN
05            TURN CN REPL MODE BIT
05            DC DELARRAY - DELETE OLD ARRAYS
05            DO ACCARRAY - ADD NEW ARRAYS
04          ELSE
05            IF TEST OPTION,THEN
06              TURN ON TEST MOCE BIT
06              TURN CN REPLACE MODE BIT
06              DO DELARRAY - DELETE OLD ARRAYS
06              CO ADDARRAY - ACD NEW ARRAYS
05            ELSE
06              PRINT, INVALID CPTION - TEST ASSUMED,MSG
06              TURN ON TEST MOCE BIT
06              TURN CN REPLACE MODE BIT
06              DO DELARRAY - DELETE OLD ARRAYS
06              DO ADCARRAY - ACD NEW ARRAYS
05            ENDIF
04          ENDIF
03        ENDIF
02      ENDIF
02      LINK TO DPPXDBAT
01    ELSE
02      PRINT NC ARRAYS DEFINED - NO PROCESSING MSG
02      TAKE ERROR EXIT - ALL
01    ENDIF
```

Figure 3-145 (2 of 7). DPPXDBAS

```
01     IF NOT TEST MODE,THEN
02       OPEN PDS READ DCB(UPDAT)
02       POINT CCB TO @INIT
02       DUMMY READ
02       MOVE IN NUM DA ARYS  DR 5203
02       WRITE NEW @INIT RECORD
01     ENDIF
     ELSE
01     PRINT UNABLE TO OPEN DATABASE PDS MSG
01     TAKE ERROR EXIT - ALL
     ENDIF
     ALL - ENTER HERE FRCM ERROR EXIT
     PRINT FPP CCMPLETED MSG
     FREE SAVE/WORK AREA
     RESTORE REGS 0 - 12
     RETURN
     BGNSEG ADCARRAY - ADD ARRAYS TO CATA BASE
01     WHILE NCT END OF CBDEF,DO
02       IF DUMMY BIT IS NOT CN,THEN
03         IF ARRAY IS BLOCKED,THEN
04           IF BLKCT IS ZERO,THEN
05             CALC  AND SAVE BLOCK CCUNT
04           ENDIF
03         ENDIF
03         IF LOGABLE ARRAY,THEN
04           LET CPPXDBLG SET UP LOGGING ARRAY
03         ENDIF
03         BLDL CN ARRAY NAME
03         IF RC=4,THEN
04           IF NOT CA RESIDENT,THEN
05             DO ADCARPDS - ACD ARRAY TO PDS
04           ELSE IT IS CA RESIDENT
05             DO ADCARDDS - ADD ARRAY TO DIRECT DATA SET
04           ENDIF
03         ELSE
04           IF RC=8,THEN
05             PRINT BLDL IO ERROR - RC=8 MSG
04           ELSE
05             IF NO DELETES DONE,THEN
06               SET DUMMY FLAG FCR FCUND ARRAY
06               PRINT DUP ARRAY NAME MSG
05             ELSE
06               IF NCT CA RESIDENT,THEN
07                 DC ADCARPDS - ADD ARRAY TO PDS
06               ELSE
07                 DO ADDARDDS - ADC ARRAY TO CIRECT CATA SET
06               ENDIF
05             ENDIF
04           ENDIF
03         ENDIF
```

Figure 3-145 (3 of 7). DPPXDBAS

```
02      ELSE
03        IF NO DELETES DCNE,THEN
04          PRINT DUMMY BIT SET MSG
03        ENDIF
02      ENDIF
02      TURN OFF FIRST ICB BIT
02      TURN CFF FIRST CATA BIT
02      INCREMENT TO NEXT ARRAY
01    ENDCO
    ENDSEG ADDARRAY          END ADD ARRAY SEGMENT
    BGNSEG ADDARPDS - ADD ARRAY TO PDS
01    DO WRITEICB - WRITE ICB'S TO PDS
01    IF ARRAY IS BLOCKED,THEN
02      DO WRITEBLK - WRITE DATA BLOCKS
01    ELSE
02      WRITEDTA - WRITE UNBLOCKED CATA
01    ENDIF
01    DO STOWDIR - STOW DUMMY DIR ENTRY FOR EOF
    ENDSEG ADDARPDS          END ADD ARRAY TO PDS SEGMENT
    BGNSEG ADDARDDS - ACD ARRAY TO DIRECT CATA SET
01    DO WRITEICB - WRITE ICB'S TO PDS
01    BALR TO DPPXDBDA TO WRITE DA ARRAY
01    DO WRITEDTA - WRITE DA CCNTRCL RECORD
01    DO STOWDIR - STCW DUMMY DIR ENTRY FOR EOF
    ENDSEG ACCARCDS   END ADD ARRAY TO DIRECT CATA SET
    BGNSEG DELARR - DELETE ARRAY SEGMENT
01    TURN ON DELETES DCNE BIT
01    GET NUMBER OF ARRAYS FOR DELETE
01    IF CELETE MCDE,THEN
02      DCUBLE TO ALLOW FOR DA LOG ARRAYS
01    ENDIF
01    GETMAIN FCR DELETE TABLE
01    CLEAR ZERO STORAGE
01    WHILE NOT END OF CBDEF,DO
02      IF CUMMY BIT NCT SET,THEN
03        BLDL CN ARRAY NAME
03        IF RC=0,THEN
04          MOVE ARRAY NAME TO DEL TABLE
04          MOVE AIC TO DEL TABLE
04          IF REPLACE CPERATION,THEN
05            SAVE OLD UPDATE LEVEL
04          ELSE
05            PRINT ARRAY DELETED MSG
04          ENDIF
04          INCR TO NEXT DEL TABLE ENTRY
03        ELSE
04          IF RC=8,THEN
05            PRINT BLDL I/O ERROR - RC=8 MSG
```

Figure 3-145 (4 of 7). DPPXDBAS

```
04         ELSE
05           PRINT ARRAY NOT FCUND MSG
04         ENDIF
03       ENDIF
02     ELSE
03       PRINT DUMMY BIT SET MSG
02     ENDIF
02     INCREMENT TO NEXT ARRAY
01   ENDDC
01   PUT ENDWORD IN DELETE TABLE
01   SORT DELETE TABLE BY ARR NAME
   ENDSEG DELARRAY        END DELETE ARRAY SEGMENT
   BGNSEG GETMNDIR - GET NEW DIRECTORY BUFFER
01   CALC BUFFER SIZE
01   GETMAIN FOR NEW DIRECTORY BUFFER
01   CLEAR STORAGE
   ENDSEG GETMNDIR        END OF GETMAIN NEW DIRECTORY
   BGNSEG WRITEIECB - WRITE ITEM CONTROL BLKS TO PDS
01   IF BLOCKED ARRAY,THEN
02     ICB START - FIRST BLOCK
02     ICB STOP - FIRST BLOCK
01   ELSE
02     ICB RECCRD STOP - ARRAY
02     ICB RECORD START - ARRAY
01   ENDIF
01   CALC LENGTH OF ICB RECORD
01   IF LEN GT DS BLKSIZE,THEN
02     UNTIL LEN LT DS BLKSIZE,DO
03       IF NOT TEST MODE,THEN
04         WRITE ICB'S
03       ENDIF
03       IF FIRST ICB BIT NOT SET,THEN
04         TURN ON FIRST ICB BIT
04         IF NOT TEST MODE,THEN
05           NOTE ICB TTR
04         ENDIF
04         SAVE ICB TTR
03       ENDIF
03       SLBTRACT DS BLKSIZE FRCM LEN
02     ENDDO
01   ENDIF
01   IF NOT TEST MODE,THEN
02     WRITE REMAINDER OF ICB'S
01   ENDIF
01   IF FIRST ICB BIT NOT SET,THEN
02     TURN ON FIRST ICB BIT
02     IF NOT TEST MODE,THEN
03       NCTE ICB TTR
02     ENDIF
```

Figure 3-145 (5 of 7). DPPXDBAS

```
02        SAVE ICB TTR
01      ENDIF
      ENDSEG WRITEICB          END WRITE ICB'S
      BGNSEG WRITEDTA - WRITE UNBLOCKED CATA RECORDS
01      IF CATA LEN GT DS BKSZ,THEN
02        UNTIL CATA LEN LE DS BKSZ,DO
03          IF NOT TEST MCDE,THEN
04            WRITE DATA TO PDS
03          ENDIF
03          IF FIRST CATA BIT NOT SET,THEN
04            TURN ON FIRST DATA BIT
04            IF NOT TEST MODE,THEN
05              NCTE DATA TTR
04            ENDIF
04            SAVE CATA TTR
03          ENDIF
03          SUBTRACT DS BKSZ FRCM DATA LEN
02        ENDDO
01      ENDIF
01      IF NOT TEST MODE,THEN
02        WRITE REMAINDER OF CATA
01      ENDIF
01      IF FIRST DATA BIT NOT SET
02        TURN ON FIRST CATA BIT
02        IF NOT TEST MODE,THEN
03          NOTE CATA TTR
02        ENDIF
02        SAVE CATA TTR
01      ENDIF
      ENDSEG WRITEDTA          END WRITE UNBLOCKED DATA
      BGNSEG WRITEBLK - WRITE BLOCKED DATA RECORDS
01      IF BLKSIZE NE ZERC,THEN
02        PICK UP ARRAY BLKSIZE
01      ELSE
02        CALC BLCCK SIZE
01      ENDIF
01      IF ARR BKSZ GT DS BKSZ,THEN
02        PRINT ARRAY BLOCK SIZE TRUNCATEC MSG
01      ENDIF
01      CLEAR DATA BLCCK BUFFER
01      IF LOGABLE ARRAY,THEN
02        MCVE LOG HDR TO OUTPUT BUFFER
01      ENDIF
01      UNTIL ALL BLCCKS WRITTEN,DO
02        CLEAR OUTPUT BUFFER
02        GET LAST BLCCK NUMBER USED
02        ADD 1
02        GET START BLOCK NUMBER
02        IF NO START BLOCK NUMBER, THEN
03          USE NEXT AVAILABLE BLOCK NUMB
02        ENDIF
```

Figure 3-145 (6 of 7). DPPXDBAS

```
02       IF ST BKNO GT LAST BKNO,THEN
03         GET DIFFERENCE
03         UNTIL DIFFERENCE EQ ZERO,DO
04           DO MVDTABLK - MOVE DATA TO OUTPUT
03         ENDDO
02       ENDIF
02       IF CATA SIZE GT BLOCK SIZE,THEN
03         PRINT CATA TRUNCATED MSG
02       ENDIF
02       MCVE DATA TC BLOCK BUFFER
02       DC MVDTABLK - MOVE DATA TO OUTPUT
02       IF NO START BLOCK NUMBER,THEN
03         USE NEXT AVAILABLE BLOCK NUMBER
02       ELSE
03         IF STRT NUM LE LAST NUM,THEN
04           SET TEST MODE
04           PRINT BLK NUMBER ERROR MSG
04           USE NEXT ABAILABLE BLOCK NUMBER
03         ENDIF
02       ENDIF
02       IF NC STCP BLOCK NUMBER
03         USE STRT OR NEXT AVAILABLE BLK NUM
02       ENDIF
02       IF BLOCK IS REPEATED,THEN
03         GET REPEAT COUNT
03         UNTIL REPEAT CNT EC ZERC,THEN
04           DO MVDTABLK - MOVE DATA TO OUTPUT
03         ENDDO
02       ENDIF
02       INCR TO NEXT BLOCK CNTL BLOCK
01     ENDDO
01     CLEAR OUTPUT BUFFER
01     IF BKCT GT LAST BKNO,THEN
02       GET DIFFERENCE
02       UNTIL DIFFERENCE EQ ZERO,DO
03         CO MVDCTABLK - MOVE CATA BLK TO OUTPUT
02       ENDDO
01     ELSE
02       IF BKCT LT LAST BKNO,THEN
03         PRINT ARRAY BLOCK CCUNT EXCEEDEC MSG
02       ENDIF
01     ENDIF
01     IF ANY CATA REMAINING,THEN
02       DO WRDTABLK - WRITE DATA BLOCK
01     ENDIF
    ENDSEG WRITEBLK        END WRITE BLOCKED DATA RECORDS
    BGNSEG MVDTABLK - MCVE DATA BLOCKS TO OUTPUT
```

Figure 3-145 (7 of 7). DPPXDBAS

```
01    UNTIL PART BLK BIT IS ZERO,DO
02      IF PART BLOCK BIT IS ONE,THEN
03        GET A(UNMCVED PART)
03        GET SIZE OF UNMOVED PART
02      ENDIF
02      GET REMAINING LENGTH IN OUTPUT BUFFER
02      IF DATA WILL FIT IN OUTPUT,THEN
03        TURN OFF PART BLOCK BIT
02      ELSE
03        TURN ON PART BLOCK BIT
02      ENDIF
02      MOVE DATA TO CUTPUT BUFFER
02      IF BUFFER FULL,THEN
03        DC WRDTABLK - WRITE CATA BLK TO OUTPT
03        CLEAR CUTPUT BUFFER
02      ENDIF
01    ENDDC
    ENDSEG MVCTABLK        END MOVE DATA TO OUTPUT
    BGNSEG WRDTABLK - WRITE A BLOCK OF DATA
01    IF NOT TEST MODE,THEN
02      WRITE DATA
01    ENDIF
01    IF FIRST CATA BIT NOT SET,THEN
02      TURN ON FIRST CATA BIT
02      IF NOT TEST MODE,THEN
03        NCTE CATA TTR
02      ENDIF
02      SAVE TTR
01    ENDIF
    ENDSEG WRDTABLK        END WRITE CATA BLOCK SEGMENT
    BGNSEG STOWDIR - STOW DIRECTORY ENTRY SEGMENT
01    IF REPLACE MODE,THEN
02      IF TEST MODE,THEN
03        PRINT ARRAY TESTED MSG
02      ELSE
03        PRINT ARRAY REPLACEC MSG
02      ENDIF
01    ELSE
02      PRINT ARRAY ADDEC MSG
01    ENDIF
01    WHILE NCT END OF NEW DIR TBLE,DO
02      IF CUPE NAME,THEN
03        TURN CN TEST MODE BIT
03        PRINT CUPE ARRAY NAME MSG
02      ENDIF
01    ENDDO
01    IF NOT TEST MCDE,THEN
02      STOW DUMMY DIR ENTRY FOR EOF
01    ENDIF
    ENDSEG STOWCIR  END DIRECTORT STOW SEGMENT
```

Figure 3-146 (1 of 9). DPPXDBAT

```
    DPPXDBAT - DATABASE FPP SECOND LOAD
    SAVE REGS
    POINT TO SAVE/WORK AREA
    DO UPD@INIT - UPDATE @INIT ARRAY
    DO GETMAIC - GETMAIN FOR AIC TABLE
    DO UPDAID - UPDATE AID TABLE
    DO READDIR - READ AND SAVE DIRECTORY
    DO UPREFRSH - UPDATE @REFRSH ARRAY
    DO GETMNCID - GETMAIN FOR NEW @CIDS ARRAY
    IF - IF NOT DELETE OPERATION,THEN
01    DO NEWCIDS - BUILD TABLE OF NEW ITEM NAMES
    ENDIF
    DO UPDCIDS - UPDATE @CIDS ARRAY
    CLOSE DBINIT
    DO UPDDIR - UPDATE PDS DIRECTORY
    ALL - ENTER HERE FROM ERROR EXIT
    RESTORE REGS 0 - 12
    RETURN
    BGNSEG UPD@INIT - UPDATE @INIT ARRAY SEGMENT
01    BLDL ON @INIT ARRAY NAME
01    IF RC NE ZERO,THEN
02      BUILD NEW @INIT ARRAY
02      IF NOT TEST MODE,THEN
03        WRITE NEW @INIT ARRAY
03        SAVE TTR
02      ENDIF
02      BUILD DIR ENTRY WITH DUMMY NAME
02      IF NOT TEST MODE,THEN
03        STOW DUMMY DIR ENTRY FOR EOF
02      ENDIF
02      SAVE NEW DIR ENTRY IN TABLE
02      INDICATE NEW PDS
01    ELSE
02      UPDATE OLD @INIT DATA RECORD
02      GET @INIT TTR
02      POINT DCB TO @INIT
02      READ @INIT
02      CALC NEW TOTAL NUM OF ARRAYS
01    ENDIF
01    MOVE @CIDS ENTRY IN TO NEW DIR TABLE
01    MVE @REFRSH ENT IN NEW DIR TBLE
01    IF NOT NEW PDS,THEN
02      INCR DEL COUNT FOR @CIDS & @REFRSH
01    ENDIF
01    IF NOT TEST MODE,THEN
02      STOW DELETE DUMMY NAME
01    ENDIF
```

Figure 3-146 (2 of 9). DPPXDBAT

```
   ENDSEG UPD@INIT   END OF UPCATE @INIT SEGMENT
   BGNSEG UPDAID - UPDATE AID TABLE
01   SORT NEW DIRECTORY TABLE BY ARRAY NAME
01   OPEN DIR READ DCB
01   UNTIL END OF NEW DIR TABLE,DO
02     UNTIL END OF CIRECTORY,CO
03       IF DIREND BIT NOT CNE,THEN
04         REAC A DIRECTORY BLOCK
04         CCUNT SAVE NUMBER OF DIRECTCRY BLOCKS
03       ENDIF
03       UNTIL ENC OF DIR BLOCK,AND
03       UNTIL INCR DIR BIT IS CNE,DO
04         IF INCR DIR BIT IS CNE,THEN
05           TURN INCR DIR BIT OFF
05           INCR TC NEXT DIR ENTRY
04         ENDIF
04         IF NEW CIR LT OLD DIR ENTRY,THEN
05           IF ARRAY NOT NUMBERED,THEN
06             INCR AID COUNT BY 1
05           ELSE
06             SAVE ARRAY NUMBER AS AID COUNT
05           ENDIF
05           PICK UP CLD AID
05           GET NEW AID
05           SAVE NEW AID IN AID TABLE
05           IF LOGABLE ARRAY,THEN
06             CCUNT LCGABLE ARRAYS
05           ENDIF
05           INCR TO NEXT NEW DIR ENTRY
04         ELSE
05           IF NOT @CIDS ENTRY,AND
05           IF NOT @REFRSH ENTRY,THEN
06             IF OLD DIR IN DEL TAB,THEN
07               INCR TC NEXT DELET TABLE ENTRY
06             ELSE
07               IF NCT NUMBERED ARRAY,THEN
08                 INCR AID COUNT BY 1
07               ELSE
08                 SAVE ARRAY NUMBER AS AID COUNT
07               ENDIF
07               PICK UP OLC AID
07               GET NEW AID
07               SAVE  NEW AID TO TABLE
07               IF LOGABLE ARRAY,THEN
08                 COUNT LCGABLE ARRAYS
07               ENDIF
06             ENDIF
05           ENDIF
```

Figure 3-146 (3 of 9). DPPXDBAT

```
05             TURN ON INCR DIR BIT
04           ENDIF
04           COUNT NEW AND OLD DIR ENTRIES
03         ENDDO
03         TURN OFF INCR DIT BIT
02       ENDDO
02       TURN ON DIR END BIT
01     ENDDO
01     UNTIL END-OF-FILE ON DIR,DO
02       READ DIR BLOCK
02       COUNT DIR BLOCKS READ
02       DIREOF - ENTER HERE FROM DIR EODAD ROUTINE
01     ENDDO
01     CLOSE DIR DCB
     ENDSEG UPDAID      END OF AID UPDATE SEGMENT
     BGNSEG UPREFRSH - UPDATE REFRSH ARRAY
01     GETMAIN NEW REFRESH ARRAY AREA
01     CLEAR AREA
01     IF NOT NEW PDS,THEN
02       OPEN DBINIT2(INPUT)
02       BLDL ON @REFRSH
02       IF RC=0,THEN
03         POINT DCB TO CA CNTL RECORD
03         READ CA CNTL FOR @REFRSH
03         UNTIL ALL BLOCKS READ,DO
04           IF SEC TRK RD BIT IS NONE,THEN
05             GET NUM BLOCKS ON FIRST TRACK
04           ELSE
05             GET NUM BLOCKS ON OTHER TRACKS
04           ENDIF
04           UNTIL END OF TRACK,DO
05             RD REF ARRAY DATA
05             IF FIRST READ BIT IS ZERO,THEN
06               MOVE ENTRY ZERO TO NEW TABLE
06               TURN FIRST READ BIT ON
06               INCR TO NEXT TABLE ENTRY
05             ENDIF
05             WHILE NOT END OF BLOCK READ,DO
06               GET OLD AID
06               CALC NEW AID
06               IF NEW AID NE ZERO,THEN
07                 SAVE NEW AID IN TABLE
07                 SAVE REFRSH BLK IN TABLE
07                 INCR TO NEXT TABLE ENTRY
06               ENDIF
06               INCR TO NEXT DATA ENTRY
05             ENDDO
```

Figure 3-146 (4 of 9). DPPXDBAT

```
05              INCR TO NEXT RECORD
05              IF ALL BLKS ON TRK READ IN,THEN
06                FORCE EXIT FRCM LOOP
05              ENDIF
04            ENDDO
04            TURN ON SECCND TRK REAC BIT
04            INCR TO NEXT TRACK
03          ENDDO
02        ELSE
03          DO UPREDIRO - UPDATE REFRSH FROM OLD DIR
02        ENDIF
02        CLOSE DBINIT2
01      ENDIF
01      IF NUM LOGABLE AR NE ZERO,THEN
02        WHILE NT ED OF NW DIR EN,THEN
03          IF LOGABLE ARRAY,TFEN
04            GET NEW AID
04            SAVE AID IN REFRSH TABLE
04            INCR TO NEXT REFRSH TABLE ENTRY
03          ENDIF
03          INCR TO NEXT NEW DIR ENTRY
02        ENDDO
01      ENDIF
01      SORT REFRSH TABLE BY AID
01      WRTE CUMMY ICB REC FOR REF
01      NOTE TTR
01      TURN CN WRITE CICS BIT
01      UNTIL END CF REFRSH TABLE
02        MCVE RECORD TC WRITE BUFFER
02        CCUNT NUMBER CF CATA BLCCKS
02        BALR TC DPPXOBCA TC WRITE DATA
02        INCR TO NEXT CATA RECORD
01      ENDDO
01      TURN ON END CIC BIT
01      BALR TO DPPXOBDA TO WRITE END RECORD
01      TURN CFF WRITE CICS BIT
01      TURN OFF ENC CIDS BIT
01      IF NCT TEST MODE,THEN
02        WRITE CA CNTL FCR @REFRSH
02        SAVE DA CNTL RECORD TTR
01      ENDIF
01      BUILD DIR ENTRY WITH DUMMY NAME
01      IF NOT TEST MODE,THEN
02        STOW DUMMY CIR ENTRY FOR ECF
01      ENDIF
     ENDSEG UPREFRSH          END UPDATE REFRSH ARRAY SEGMENT
     BGNSEG UPREDIRC - UPDTE REFRSH ARRAY FROM OLD DIR
```

Figure 3-146 (5 of 9) DPPXDBAT

```
01    IF OLD DIR READ IN,THEN
02      WHILE NOT END CF DIRECTORY
03        IF LOGABLE ARRAY,THEN
04          GET OLD AID
04          CALC NEW AID
04          IF NEW AID NE ZERO,THEN
05            SAVE AID IN REFRSH TABLE
05            INCR TO NXT REFRSH TABLE ENTRY
04          ENDIF
03        ENDIF
03        INCR TO NXT CIRECTORY ENTRY
02      ENCCO
01    ENCIF
   ENDSEG UPRECIRO           END UPDATE REFRSH FROM OLD DIRECTORY
   BGNSEG NEWCIDS - BUILD NEW CIDS SEGMENT
01    IF A(NEW CIC AREA) NE ZERO,THEN
02      WHILE NCT END CF DBDEF,DO
03        IF DUMMY BIT NOT ON,THEN
04          IF ARRAY IS BLOCKED,THEN
05            GET A(ICB START) FOR FIRST BLOCK
05            GET A(ICB STOP) FOR FIRST BLOCK
04          ELSE
05            GET A(ICB START) FOR ARRAY
05            GET A(ICB STOP) FOR ARRAY
04          ENDIF
04          CALC LENGTH OF MOVE
04          MOVE ICB'S TO NEW CID TABLE
04          INCR TO START OF NEXT MCVE
03        ENDIF
03        INCREMENT TO NEXT ARRAY
02      ENDDO
02      PUT ENDWORD IN NEW CID TABLE
02      SORT NEW CIC TABLE BY ARRAY NAME
01    ENDIF
   ENDSEG NEWCIDS            END OF NEW CIDS BUILD
   BGNSEG UPDCIDS - UPCATE @CIDS ARRAY SEGMENT
01    IF NOT NEW PDS,THEN
02      BLDL CN @CICS
02      PCINT DCB TC @CIDS
01    ENDIF
01    DO UPDCIDLP - UPDATE CIDS LCCP
01    IF NOT TEST MODE,THEN
02      WRTE LST DATA RECORD TO PDS
01    ENDIF
01    IF NOT FIRST WRITE,THEN
02      TURN ON FIRST WRITE BIT
02      IF NOT TEST MCDE,THEN
```

Figure 3-146 (6 of 9). DPPXDBAT

```
03        NCTE TTR
02      ENDIF
02      SAVE CIDS TTR
01    ENDIF
01    INCR CIDS BLK CCUNT
01    TURN ON WRITE CICS BIT
01    TURN OFF END CIDS BIT
01    BALR TO DPPXCBDA TO WRITE LAST DATA
01    TURN ON ENC CID BIT
01    BALR TO DPPXDBCA TO WRITE END RECORD
01    IF NOT TEST MODE,THEN
02      WRITE CA CNTL RECORD
02      SAVE CIDS DA CNTL TTR
01    ENDIF
01    BUILD DUMMY DIR ENTRY FOR CIDS
01    IF NOT TEST MODE,THEN
02      STOW DUMMY CIR ENTRY FOR EOF
01    ENDIF
01    IF NOT TEST MODE,THEN
02      STOW DELETE CUMMY NAME
01    ENDIF
    ENDSEG UPCCIDS              END UPDATE @CIDS
    BGNSEG UPDCIDLP - CIDS UPDATE LOOP SEGMENT
01    WHILE NOT END CF OLD CIDS,OR
01    WHILE NOT END OF NEW CIDS,DO
02      IF FIRST CID BIT NCT SET,THEN
03        TURN ON FIRST CID BIT
03        GET A(END OF BUFFER)
02      ENDIF
02      IF AT END OF BUFFER,THEN
03        GET A(START OF BUFFER)
03        IF NOT NEW PDS,THEN
04          READ OLC CID RECORD
03        ELSE
04          PUT ENDWORD IN READ BUFFER
03        ENDIF
02      ENDIF
02      IF NW CID EN LT OLD CID EN,THEN
03        MCVE ENTRY TO OUTPUT BUFFER
03        INCR TO NXT NEW CID ENTRY
02      ELSE
03        MCVE ENTRY TO OUTPUT BUFFER
03        IF NOT ENDWORC,THEN  CR#5410
04          INCR TC NXT CLD CID ENTRY
03        ENDIF DR#5410
02      ENDIF
02      PICK UP OLC AID
02      CALC NEW AID
```

Figure 3-146 (7 of 9). DPPXDBAT

```
02      IF NEW AID NE ZERO,THEN
03        SAVE AID IN MOVED ENTRY
03        IF ITEM NME NE BLANKS,THEN
04          IF ITEM NME ID DUP,THEN
05            PRINT CUP ITEM NAME MSG
05            SET TEST MODE
04          ENDIF
04          SAVE TO PREV NAME
04          INCR TC NXT SLOT IN OUTPUT BUFFER
03        ENDIF
02      ENDIF
02      IF END OF OUTPUT BUFFER,THEN
03        IF NCT TEST MODE,THEN
04          WRITE OUTPUT BUFFER TO PDS
03        ENDIF
03        IF FIRST ICB BIT NOT SET,THEN
04          TURN ON FIRST ICB BIT
04          IF NOT TEST MODE,THEN
05            NOTE TTR
04          ENDIF
04          SAVE CIDS TTR
03        ENDIF
03        TURN CN WRITE CIDS BIT
03        INCR CICS BLK COUNT
03        BALR TO DPPXCBDA TO WRITE TO DDS
02      ENDIF
01    ENDCO
   ENDSEG UPDCIDLP          END OF CIDS UPDATE LOOP
   BGNSEG GETMAIC - GETMAIN FOR AID TABLE
01    CALC SIZE OF GETMAIN
01    GETMAIN AREA FOR AID TABLE
01    CLEAR GETMAIN AREA
   ENDSEG GETMAID           END OF AID TABLE GETMAIN
   BGNSEG GETMNCID - GETMAIN FOR NEW CID TABLE
01    IF NCT DELETE MODE
02      CALC SIZE OF GETMAIN
02      GETMAIN FOR NEW CID TABLE
02      CLEAR GETMAIN AREA
01    ELSE
02      DUMMY OUT NEW CID TABLE
01    ENDIF
   ENCSEG GETMNCID          END OF GETMAIN FOR NEW CIC TABLE
   BGNSEG READDIR - READ PDS DIRECTORY SEGMENT
01    IF
01    ENDIF
01    IF
01    ENDIF
```

Figure 3-146 (8 of 9). DPPXDBAT

```
01    IF NM DIR BLK ND GT NM DIR BLKS,THEN
02      PRINT INSUFFICIENT DIRECTORY SPACE MSG
02      SET TEST MODE
01    ELSE
02      GETMAIN FOR DIR INPUT BUFFER
02      CLEAR BUFFER
02      GETMAIN FOR DECB BUFFER
02      CLEAR BUFFERE
02      GETMAIN FOR DIR OUTPUT BUFFER
02      CLEAR BUFFER
02      OPEN DIR READ DCB(INPUT)
02      WHILE NOT END OF OUTPUT BUFFER,DO
03        CHAIN OUTPUT DIR BLOCKS
02      ENDDO
02      UNTIL ALL DIR BLKS READ,DO
03        READ DIR BLOCK
03        MOVE DIR ENTRIES TO INPUT BUFFER
03        INCR TO NEXT SPACE
02      ENDDO
02      CLOSE DIR READ DCB
01    ENDIF
    ENDSEG READDIR            END OF DIRECTORY READ SEGMENT
    BGNSEG UPDIR - UPDATE PDS DIRECTORY
01    IF NOT TEST MODE,THEN
02      DO UPDDIRLP - UPDATE DIRECTORY LOOP
02      OPEN DIR DCB(UPDATE)
02      UNTIL END OF DIR OUTPUT BUFFER,DO
03        DUMMY READ FOR UPDATE MODE
03        MOVE DIR BLOCK TO OUTPUT AREA
03        IF NOT TEST MODE,THEN
04          WRITE DIRECTORY BLOCK
03        ENDIF
03        INCR TO NEXT DIR OUTPUT BLOCK
02      ENDDO
01    ENDIF
01    DO FREETABL - FREEMAIN CONTROL TABLE
    ENDSEG UPDDIR             END OF DIRECTORY UPDATE SEGMENT
    BGNSEG UPDDIRLP - DIRECTORY UPDATE LOOP SEG
01    IF NEW PDS,THEN
02      DUMMY OUT OLD DIRECTORY
01    ENDIF
01    WHILE NOT END OF OLD DIR,OR
01    WHILE NOT END OF NEW DIR,DO
02      IF NEW DIR LT OLD DIR,THEN
03        MOVE IN NEW DIR ENTRY
03        INCR TO NEXT ENTRY
02      ELSE
03        MOVE IN OLD DIR ENTRY
03        INCR TO NEXT ENTRY
02      ENDIF
```

Figure 3-146 (9 of 9). DPPXDBAT

```
02      GET OLD AID
02      CALC NEW AID
02      IF NEW AID NE ZERO,THEN
03        INCR DIR COUNTER
03        SAVE NEW AID IN DIR ENTRY
03        IF DA ARRAY,THEN
04          INCR DA ARRAY COUNTER
03        ENDIF
03        IF ENTRY IS @CIDS,THEN
04          MOVE IN REAL DIR ENTRY
03        ELSE
04          IF ENTRY IS @REFRSH,THEN
05            MOVE IN REAL DIR ENTRY
04          ENDIF
03        ENDIF
03        IF NEW DIR BLOCK NOT FULL,THEN
04          INCR TO NEXT OUTPUT DIR ENTRY
03        ELSE
04          MOVE LAST NAME TO KEY
04          SET LENGTH
04          INCR TO A(NEXT DIR OUTPUT BLOCK)
04          INCR TO FIRST DIR ENTRY THIS BLOCK
04          DIR COUNTER TO ZERO
03        ENDIF
02      ENDIF
01    ENDDO
01    MOVE ENDWORD TO LAST NAME
01    MOVE ENDWORD TO KEY
    ENDSEG UPDDIRLP                 END DIRECTORY UPDATE LOOP
    BGNSEG FREETABL - TABLE FREEMAIN SEGMENT
01    FREE DECB BUFFER
01    FREE DIR INPUT BUFFER
01    IF ARRAYS WERE DELETED,THEN
02      FREE DELETE TABLE
01    ENDIF
01    IF NOT DELETE MODE,THEN
02      FREE NEW CID TABLE
01    ENDIF
01    FREE AID TABLE
01    FREE NEW DIRECTORY BUFFER
    ENDSEG FREETABL
```

Figure 3-147 (1 of 5). DPPXDBCP

```
    DPPXDBCP - DATABASE BDAM DATA SET COMPRESS
    SAVE REGS
    GET SAVE/WORK AREA
    OPEN STSPRINT AND CBINIT(INPUT)
    IF OPEN
01    SET RC=16
01    IF OPEN
02      PRINT CCMPRESS STARTED MESSAGE
02      BLDL FOR @INIT
02      IF RC=0,THEN
03        DO CNTLTABL - GET CCNTROL TABLE STORAGE
03        CLOSE DBINIT
03        DO READDIR - READ DIRECTORY ENTRIES
03        DO READCNTL - READ A CCNTROL RECORDS
03        DO SORTTABL - SORT CONTRCL TABLE
03        CEN DBINIT(UPDAT)
03        DO MOVARAYS - MOVE ARRAYS
02      ELSE
03        IF RC=4,THEN
04          PRINT INVALID DATABASE DATA SET MSG
03        ELSE
04          PRINT RC=8 FROM BLDL - PERM I/O ERROR MSG
03        ENDIF
02      ENDIF
02      PRINT END OF CCMPRESS MSG
01    ELSE
02      PRINT UNABLE TO OPEN CBINIT MSG
01    ENDIF
    ENDIF
    CLOSE CBINIT AND SYSPRINT
    FREE CCNTRCL TABLE
    LINK TO DPPXDBIN        DR#2343
    FREE SAVE/WORK AREA
    RESTORE REGS 0 - 12
    RETURN
    BGNSEG CNTLTABL - GET CONTRCL TABLE STORAGE
01    GET @INIT TTR
01    POINT TO @INIT
01    READ IN @INIT
01    GET NUMBER OF DA ARRAYS IN DATA BASE
01    IF NUM ARRAYS EQ ZERO,THEN
02      PRINT NO DIRECT ACCESS ARRAYS IN CATABASE MSG
02      SET RC=4 - NO DA ARRAYS IN DATABASE
01    ENDIF
01    CALC LENGTH OF CNTL TABLE
01    GETMAIN FOR TABLE
01    CLEAR CNTL TABLE BUFFER
    ENDSEG CNTLTABL  END CNTL TABLE GETMAIN RCUTINE
    BGNSEG READDIR - READ DIRECTORY BLOCKS
```

Figure 3-147 (2 of 5). DPPXDBCP

```
01     OPEN CBINIT FOR DIR READ
01     UNTIL END OF DIRECTORY,DO
02       REAC A CIRECTCRY BLOCK
02       TURN OFF INCR CIR BIT
02       UNTIL END OF DIR BLOCK,CO
03         IF INCR CIR BIT IS CNE,THEN
04           INCR TO NEXT DIR ENTRY
03         ELSE
04           TURN ON INCR DIR BIT
03         ENDIF
03         IF NOT END OF DIRECTORY,TFEN
04           IF DA ARRAY,THEN
05             GET CNTL RECORD TTR
05             GET BLOCK COUNT
05             GET BLOCK SIZE
05             INCR TC NEXT CNTL TABLE ENTRY
04           ENDIF
03         ENDIF
02       ENDDO
01     ENDCO
01     DIREOF - ENTRY FRCM EODAD CN DIR READ
01     PUT ENDWORD AT END OF TABLE
01     SAVE A(ENCWORD IN CNTL TABLE)
01     TURN OFF INCR DIR BIT
01     CLOSE DBINIT FOR CIR READ
     ENDSEG READCIR   END DIRECTORY REAC SEGMENT
     BGNSEG READCNTL - READ DA CCNTROL RECORDS
01     OPEN CBINIT FOR INPUT
01     UNTIL END CF CNTL TABLE
02       GET A(CNTL RECORD TTR)
02       PCINT TO CNTL RECORD
02       READ DA CNTL RECORD
02       INCR TO NEXT CNTLTABLE ENTRY
01     ENDDO
01     CLOSE DBINIT2
     ENDSEG READCNTL  END READ CNTL RECORD SEGMENT
     BGNSEG SORTTABL - SORT DA CNTL TABLE
01     GET A(CNTL TABLE)
01     GET SIZE CF RECCRDS
01     GET A(LAST CNTL TABLE ENTRY)
01     SORT THE CCNTROL TABLE BY ADDNAME
     ENDSEG SORTTABL          END CF SORT DA CNTL TABLE SEGMENT
     BGNSEG MOVARAY - MOVE DIRECT ACCESS ARRAYS
01     UNTIL END OF CNTL TABLE,DO
02       MOVE CURRENT CADD TO PREVIOUS
02       MOVE DADD TO DCB
02       OPEN DADC(INPUT)
```

Figure 3-147 (3 of 5). DPPXDBCP

```
02     UNTIL DADD NE PREVIOUS,DO
03       IF DCB OP,THEN
04         SAVE DDS BLKSIZE
04         CPEN SYSUT1(OUTPUT)
04         IF OPEN,THEN
05           DO READWRIT - READ DADD/WRITE TO SYSUT1
05           SET READ/WRITE BIT
04         ELSE
05           PRINT UNABLE TO OPEN SYSUT1 MSG
05           SET RC=8 - UNABLE TO OPEN SYSUT1
05           TAKE ERROR EXIT - ALL
04         ENDIF
03       ELSE
04         IF FIRST SKIP FOR DADD,THEN
05           SET FIRST SKIP BIT
05           PRINT NO DD STATEMENT FOR DDS MSG
04         ENDIF
03       ENDIF
03       INCR TO NEXT CNTL TABLE ENTRY
02     ENDDO
02     TURN OFF FIRST SKIP BIT
02     IF READ/WRITE BIT IS ONE,THEN
03       DO WRDDSEND - WRITE DDS END RECORD
03       CLOSE DADD AND SYSUT1
03       UNTIL
03       ENDDO
03       PUT SYSUT1 DDNAME BACK IN DCB
02     ENDIF
01   ENDDC
   ENDSEG MOVARAYS        END MOVE ARRAY SEGMENT
   BGNSEG READ/WRIT - READ/WRITE CA CATA RECORDS
01   TURN OFF FIRST CATA BIT
01   TURN CN FIRST TRK BIT
01   TURN OFF SECOND TRK BIT
01   UNTIL NUM WRITES EQ BLKCT,DC
02     IF SECCND TRK BIT IS OFF,THEN
03       GET NUM BLOCKS CN FIRST TRACK
02     ELSE
03       CET NUM BLOCKS CN OTHER TRACKS
02     ENDIF
02     UNTIL END CF TRK,DO
03       READ FROM DADD
03       INCR TO NEXT RECORD
03       DO WRITEDDS - WRITE RECORD TO SYSUT1
03       INCR RECORD COUNTER
03       IF END OF ARRAY,THEN
04         FORCE EXIT FROM LOOP
03       ENDIF
02     ENDDO
```

Figure 3-147 (4 of 5). DPPXDBCP

```
02     TURN CN SECCND TRK READ
02     INCR TO NEXT TRACK
02     SET RECORD NUMBER TO ONE
01   ENDDO
01   IF FIRST TRK BIT IS CNE,THEN
02     SAVE NUM BLOCKS CN TRACK ONE
02     TURN OFF FIRST TRACK BIT
01   ELSE
02     IF SECOND TRK BIT IS ONE,THEN
03       SAVE NUM BLCCKS CN SECOND TRACK
03       TURN OFF SECOND TRK BIT
02     ENDIF
01   ENDIF
01   TURN OFF SECCND TRK READ
01   MOVE NEW CATA TO CNTL TABLE
   ENDSEG READWRIT        END READ/WRITE CNTL SEGMENT
   BGNSEG WRITEDDS - WRITE DIRECT CATA SET
01   IF END CATA BIT IS ONE,THEN
02     GET MAX DATA SET BLKSIZE
01   ELSE
02     GET ARRAY BLKSIZE
01   ENDIF
01   UNTIL GOOD CATA WRITE,DO
02     WRITE A RECCRD
02     IF RC=4,THEN
03       UNTIL GC CAPACITY REC WRTE,THEN
04         WRITE CAPACITY RECORD
04         IF RC NE 12,THEN
05           SET GOOD CAPACITY RECORD WRITE BIT
05           IF END DATA BIT IS ONE,THEN
06             TURN ON END TRACK BIT
06             TURN CN GOOD CATA WRITE BIT
05           ELSE
06             IF FIRST TRK BIT IS ONE,THEN
07               IF BLK PER TRK NOT ZERO,THEN
08                 SAVE NUM BLCCKS CN TRACK CNE
08                 TURN OFF FIRST TRACK BIT
08                 TURN ON SECOND TRACK BIT
07               ENDIF
06             ELSE
07               IF SECOND TRK BIT IS ONE,THEN
08                 SAVE NUM BLOCKS ON SECOND TRACK
08                 TURN OFF SECOND TRACK BIT
07               ENDIF
06             ENDIF
C5           ENDIF
04         ENDIF
03       ENDDO
```

Figure 3-147 (5 of 5). DPPXDBCP

```
03        TURN CFF GOCD CAPACITY WRITE BIT
02      ELSE
03        IF RC NE 12,THEN
04          TURN CN GOCD DATA WRITE BIT
04          INCR BLOCK PER TRACK COUNTER
03        ENDIF
02      ENDIF
01    FNDDC
01    TURN OFF GOOD CATA WRITE BIT
01    IF FIRST CATA BIT IS ZERO,THEN
02      GET A(MBBCCHHR)
02      LINK TO CCNVERSICN ROUTINE
02      SAVE FIRST RECORD TTR
02      TURN ON FIRST CATA BIT
01    ENDIF
    ENDSEG WRITEDDS           END OF DDS CATA WRITE SEGMENT
    BGNSEG WRITCNTL - WRITE NEW CA CONTROL RECORD
01    POINT TO DA CNTL RECORD
01    DUMMY REAC IN UPDAT MODE
01    MVE NEW CA CNTL REC INTO BUFFER
01    WRITE NEW DA CNTL RECORD
    ENDSEG WRITCNTL           END OF CA CNTL RECORD SEGMENT
    BGNSEG WRDDSEND - WRITE END OF DDS
01    UNTIL GOOD CAPACITY REC WRTE,DO
02      WRITE CAPACITY RECORD
02      IF RC NOT EC 12,THEN
03        SET GCOD CAPACITY RECORD WRITE BIT
03        IF FIRST TRK BIT IS ONE,THEN
04          SAVE NUM BLCCKS CN FIRST TRACK
04          TURN OFF FIRST TRK BIT
04          TURN ON SECCND TRK BIT
03        ELSE
04          IF SECOND TRK BIT IS ONE,THEN
05            SAVE NUM BLOCKS ON SECOND TRACK
05            TURN OFF SECCND TRK BIT
04          ENDIF
03        ENDIF
02      ENDIF
01    ENDDC
01    TURN OFF GCOD CAPACITY REC WRTE BIT
01    CLEAR BUFFER
01    SET END OF CATA BIT
01    UNTIL END CF TRK,CO
02      DO WRITEDDS - WRITE DDS DATA
01    ENDDC
01    TURN OFF END CF DATA BIT
01    TURN OFF END CF TRK BIT
01    TURN OFF FST CATA BIT
    ENDSEG WRDDSEND
```

Figure 3-148 (1 of 5). DPPXDBDA

```
    DPPXDBUA - CATA BASE DIRECT ACCESS FPP
    SAVE REGS
    PCINT TO SAVE/WORK AREA
    IF NOT-FST-CID BIT ZERO,THEN
01    ZERC WORK AREA
    ENDIF
    IF NOT @CIDS CR @REFRSH,THEN
01    IF BLCCKED ARRAY,THEN
02      IF CADD NAME IS BLANK,THEN
03        MVE CBINIT2 DCNAME
02      ELSE
03        MOVE IN DDNAME
02      ENDIF
02      DO OPENDDS - CPEN BDAM CCB
02      DO WRDDSCTL - WRITE DDS CCNTROL SEGMENT
02      DO WRDDSEND - WRITE DDS END RECORD
02      IF BUFFER EXISTS,THEN   DR#2056
03        FREE BUFFER
02      ENCIF DR#2056
02      IF DDS DCB IS OPEN,THEN
03        CLOSE CCS DCB
03        IF BUFFERS
04          FREEPOOL BUFFERS        DR#2056
03        ENCIF CR#2056
02      ENDIF
01    ELSE
02      PRINT NOT A BLOCKED ARRAY MSG
02      ENTER TEST MODE
01    ENDIF
    ELSE
01    IF NOT-FST-CID BIT ZERO,THEN
02      MVE DBINIT2 DD NAME
02      DC CPENDDS - OPEN DDS CCB
02      SET NOT-FST-CID BIT TC ZERO
01    ENDIF
01    IF END-CID BIT IS ZERO,THEN
02      DO WRITEDDS - WRITE DDS RECORD
01    ELSE
02      DO WRDDSEND - WRITE CDS END RECORD
01    ENDIF
01    IF END-CIC BIT IS ONE,THEN
02      CLCSE DDS DCB
02      IF BUFFERS
03        FREEPCOL BUFFERS        DR#2056
02      ENDIF DR#2056
02      SET NOT-FST-CIC BIT TO ZERO
01    ENDIF
```

Figure 3-148 (2 of 5). DPPXDBDA

```
   ENDIF
   RESTORE REGS 0 - 12
   RETURN
   BGNSEG
01   DDNAME TO CA CNTL RECORD
01   OPEN DDS DCB
01   IF
02     SET TEST MODE
02     PRINT TEST MCDE DUE TO BAC OPEN MSG
02     WORK WITH DEFAULT MAX RECORC SIZE
01   ELSE
02     IF NO BLKSIZ,THEN
03       GET MAX BLKSIZE FOR DEVICE
02     ELSE
03       GET DDS BLKSIZE
02     ENDIF
01   ENDIF
01   IF WRITE-CIDS BIT IS OFF,THEN
02     GETMAIN FOR RECORD SIZE BUFFER
01   ELSE
02     SET BUFFER ADDRESS TO ZERO
01   ENDIF
01   SAVE BUFFER ADDRESS
   ENDSEG OPENDDS           OPEN DDS DCB
   BGNSEG WRDDSCTL - WRITE DDS ARRAY CTRL SEGMNT
01   IF ARRAY BLKSIZE IS NT ZERO,THEN
02     GET DATA BLOCK SIZE
01   ELSE
02     CALC BLOCK SIZE
02     PUT SIZE IN ARRAY
01   ENDIF
01   IF BKSZ GT DS BKSZ,THEN
02     TRUNCATE ARRAY BKSZ
02     PRINT ARRAY BLOCK SIZE TRUNCATED MSG
01   ENDIF
01   UNTIL END OF BLOCK CNTL BLOCKS,DO
02     CLEAR BUFFER
02     GET LAST BLOCK NUMBER USED
02     GET START BLOCK NUMBER
02     IF NO START BLOCK NUMB,THEN
03       USE NEXT AVAILABLE BLOCK NUMBER
02     ENDIF
02     IF STRT BKNO GT LAST BKNO,THEN
03       GET DIFFERENCE
03       UNTIL LAST BKNO EQ START BKNO
04         DO WRITEDDS - WRITE ZEROS TO DDS
03       ENDDO
02     ENDIF
```

Figure 3-148 (3 of 5). DPPXDBDA

```
02        IF NOT LCGGING ARRAY,THEN
03          GET SIZE CF CATA IN BLOCK
03          IF CATA SIZE GT BLOCK SIZE,THEN
04            PRINT CATA TRUNCATED MSG
04            TRUNCATE TO BLOCK SIZE
03          ENDIF
03          MOVE DATA TO CUTPUT BUFFER
02        ENDIF
02        IF NO START BLOCK NUMBER,THEN
03          USE NEXT AVAILABLE BLOCK NUMBER
02        ELSE
03          IF
04            PRINT BLK NUMBER ERROR MSG
04            USE NEXT AVAILABLE BLK NUMBER
03          ENDIF
02        ENDIF
02        IF NO STOP BLCCK NUMBER,THEN
03          USE START BLOCK NUMBER
02        ENDIF
02        IF STOP NUM+1 GE START NUM,THEN
03          UNTIL ALL DUPLICATIONS ARE DONE,CO
04            DC WRITEDDS - WRITE DATA TO DCS
03          ENDDO
02        ENDIF
02        INCR TC NEXT CCNTROL BLCCK
01      ENDDO
01      IF BKCT GT LAST BKNO,THEN
02        GET DIFFERENCE
02        IF NOT LOGGING ARRAY
03          CLEAR BUFFER
02        ENDIF
02        UNTIL ALL BLCCKS WRITTEN,DO
03          DC WRITEDDS - WRITE ZEROS TO DDS
02        ENDDO
01      ELSE
02        IF BKCT LT LAST BKNO,THEN
03          PRINT ARRAY BLOCK CCUNT EXCEEDED MSG
02        ENDIF
01      ENDIF
      ENDSEG WRDDSCTL        END OF DDS CONTROL SEGMENT
      BGNSEG WRDDSEND - WRITE DDS END RECORD
01      IF NCT IN TEST MODE,THEN
02        UNTIL CAPACITY RECORD WRITEN,DO
03          WRITE CAPACITY RECORD
03          IF RETURN CODE IS NCT 12,THEN
04            SET GOOD WRITE SZ BIT
04            IF FIRST TRK BIT IS ONE,THEN
05              SAVE BLOCKS CN TRACK CNE
05              TURN OFF FIRST TRACK BIT
05              TURN CN SECOND TRACK BIT
```

Figure 3-148 (4 of 5). DPPXDBDA

```
04          ELSE
05            IF SECOND TRK BIT IS CNE,THEN
06              SAVE BLCCKS CN SECCND TRACK
06              TURN OFF SECOND TRACK BIT
05            ENCIF
04          ENDIF
03        ENDIF
02      ENDCO
01    ENDIF
01    TURN CFF GOOD WRITE SZ BIT
01    IF NCT WRITE CIDS,THEN
02      CLEAR BUFFER
01    ENDIF
01    SET END OF CATA BIT
01    UNTIL END OF TRACK,DO
02      DO WRITEDCS - PAD TO END CF TRACK
01    ENDDO
    ENDSEG WRDDSEND END OF WRITE CCS END SEGMENT
    BGNSEG WRITEDDS - WRITE DATA TO DDS
01    IF WRITE CIDS BIT IS ONE,THEN
02      GET A(ICB BUFFER)
01    ELSE
02      GET A(DATA BUFFER)
01    ENDIF
01    IF END CATA BIT IS CNE,THEN
02      USE MAX LENGTH FCR WRITE
01    ELSE
02      IF WRITE CICS BIT IS ONE,THEN
03        USE CIDS RECORD SIZE
02      ELSE
03        USE LENGTH OF DATA
02      ENDIF
01    ENDIF
01    IF TEST MODE IS NOT SET,THEN
02      UNTIL GCOD CATA WRITE,DC
03        WRITE CATA RECORD TO CDS
03        IF RC=4,THEN
04          UNTIL GOOD CAPACITY REC WRTE,DO
05            WRITE CAPACITY RECORD
05            IF RC NOT EQUAL 12,THEN
06              SET GOOD CAPACITY RECORC BIT
06              IF END OF CATA BIT IS ONE,THEN
07                SET END OF TRK BIT
07                SET GOOD CATA WRITE BIT
06              ELSE
07                IF FIRST TRK BIT IS ONE,THEN
08                  IF BLKS PER TRK IS NOT ZERO,THEN
09                    SAVE BLOCKS CN TRACK ONE
```

Figure 3-148 (5 of 5). DPPXDBDA

```
09                    TURN OFF FIRST TRACK BIT
09                    TURN CN SECONC TRACK BIT
08                  ENDIF
07                ELSE
08                  IF SECOND TRK BIT IS ONE,THEN
09                    SAVE BLOCKS CN SECOND TRACK
09                    TURN OFF SECOND TRACK BIT
08                  ENDIF
07                ENDIF
06              ENDIF
05            ENDIF
04          ENDDO
03        ELSE
04          IF RC NOT EQUAL 12,THEN
05            SET GOOD CATA WRITE BIT
05            INCR BLCCK PER TRK CCUNTER
04          ENDIF
03        ENDIF
02      ENDDO
02      SET GOOD DATA WRITE BIT
02      IF FST DTA WRTE BIT ZERC,THEN
03        GET A(MBBCCHHR)
03        LINK TO CONVERSION ROUTINE
03        SAVE FIRST RECORD TTR
03        SET FIRST CATA WRITE BIT
02      ENDIF
01    ELSE
02      IF END OF CATA BIT IS ONE,THEN
03        SET END OF TRK BIT
02      ENCIF
01    ENDIF
   ENDSEG WRITECDS  WRITEDDS - WRITE DATA TO DCS
```

Figure 3-149. DPPXDBIN

```
    DPPXCBIN-MAIN SEGMENT
     INITIALIZE WORK AREAS
     COPY CPXDBIN6                             READ IN DIRECTORY BLOCKS
     IF NC ERRORS OCCURED
01     COPY CPXCBIN1                           OBTAIN CORE FOR @INIT DATA
01     IF NO ERRCRS OCCURED
02       COPY DPXDBIN2                         BUILD CATA BASE CONTROL
02       IF NO ERRORS OCCURED
03         COPY DPXDBIN3                       SORT VS RESIDENT ARRAY DATA
03         IF NO ERRORS OCCURED
04           COPY DPXDBIN4                     WRITE @INIT ITEM AND DATA RECORDS
03         ENDIF
02       ENDIF
01     ENDIF
     ENDIF
     IF ERRCRS OCCURED
01     ABEND STEP WITH USER CCMPLETICN CODE
    ENDIF
    ENDSEGMENT
```

Figure 3-150 (1 of 2). DPPXDBLG

```
     DPPXDBLG - CATA BASE LOGGING FPP
     SAVE REGS
     GET SAVE/WCRK AREA
     IF NOT PRINT REQUEST,THEN
01     GET A(STRT OF LOGBLE ARRY DTA -(LOG HDR))
01     GET A(LOGGING ARRAY)
01     IF CADC LCGNG IS BLANK,THEN
02       INSRT CBINIT2 DCB
01     ELSE
02       INSRT DDNME IN CCB
01     ENDIF
01     CPEN DA DDNAME DCB
01     IF
02       SET RUN TO TEST MODE
02       PRINT ERROR MESSAGE
02       WORK WITH MAX REC SIZE
01     ELSE
02       IF NO BLKSIZ,THEN
03         GET MAX BLKSIZ FOR DEVICE
02       ELSE
03         GET DCS BLKSIZE FROM CCB
02       ENDIF
01     ENDIF
01     SAVE BLKSIZE
01     IF LCGBLE ARRAY IS BLOCKC,THEN
02       IF BLKSIZE IS ZERO,THEN
03         CALCULATE BLKSIZE
02       ENDIF
02       CALCULATE TOTAL SIZE OF ARRAY
02       IF TOT LEN GT CS BLKSIZE,THEN
03         IF BLKSIZE LESS THAN LG HD SIZE
04           CALCULATE NUM BLKS TO HOLD LOG HEADER
04           CALCULATE TCTAL ARRAY SIZE
03         ELSE
04           ADD A BLOCK TO HCLD LOGHDR
03         ENDIF
02       ELSE
03         MAKE TOT LEN W/LOGHDR BE BLKSIZE
02       ENDIF
01     ELSE
02       GET ARRAY SIZE (INCLUDING LG HDR)
01     ENDIF
01     IF TOT SZ GT CS BLKSIZE,THEN
02       CALCULATE LCGGING ARRAY BLOCKS PER COPY
01     ELSE
02       DA LOG ARRAY BLKS PER COPY = 1
01     ENDIF
01     CALCULATE CA LOG ARRAY BLCCK CCUNT
```

Figure 3-150 (2 of 2). DPPXDBLG

```
01    SAVE CA LCG ARRAY BLCCK CCUNT
01    SAVE DA LCG ARRAY BLCCK SIZE
01    SAVE DA LOG ARRAY BLOCKS PER COPY
01    IF CCB CP,THEN
02      CLOSF DA CCB
02      IF EUFFERS
03        FREEPOOL BUFFERS        CR#2056
02      ENDIF DR$2056
01    ENDIF
   ELSE
01    DO PRINT LINE FROM INPUT
   ENDIF
   FREE SAVE/WORK AREA
   RESTORE REGS 0 - 12
   RESET RETURN CODE
   RETURN
   BGNSEG FCRMAT AND PRINT A LINE
01    PUT LINE TO SYSPRINT
   ENDSEG PRINT              PRINT ROUTINE
```

Figure 3-151. DPPXDEFL

```
     DPPXDEFL-DEFINE LOCK MAIN SEGMENT
01     LOAC ACDRESS OF LOCK CCNTROL BLOCK CHAIN
01     WHILE NEXT CCNTROL BLOCK ADDRESS IS NON-ZERO,AND
01     UNTIL A PREVIOUSLY DEFINED CCNTROL BLOCK IS FOUND FOR THIS RESOURCE
02       LOAD ADDRESS OF NEXT CONTRCL BLOCK
01     ENDDO
01     IF TYPE=CET REQUEST
02       IF BLOCK NCT PREVIOUSLY DEFINEC
03         GETMAIN LOCK CCNTROL BLOCK STORAGE
03         INITIALIZE CCNTROL BLOCK
03         CHAIN CONTROL BLOCK INTO CHAIN OF LOCK CCNTROL BLOCKS
02       ENCIF
02       LOAD ADDRESS CF THIS LOCK CCNTROL BLOCK
01     ELSE
02       IF  TYPE=REL REQUEST
02       IF BLOCK IS PREVIOUSLY DEFINED
03         DECREMENT USE COUNT
03         IF COUNT IS LESS THAN 1
04           REMOVE LCCK CCNTROL BLOCK FROM CHAIN OF LOCK CONTROL BLCCKS
04           FREE  CCNTROL BLOCK STORAGE
03         ENCIF
02       ENCIF
02       ELSE
03         IF TYPE=FIND REQUEST
04           IF BLCCK PREVIOUSLY DEFINED
05             LOAC ACCRESS OF THIS LOCK CONTROL BLCCK
04           ELSE
05             SET ERROR CODE
04           ENDIF
03         ENDIF
02       ENDIF
01     ENCIF
     ENDSEGMENT
```

Figure 3-152. DPPXDPB

```
    DPPXDPB MAIN SEGMENT * DATA PLAYBACK ROUTINE *
01    LOCK DATA PLAYBACK ROUTINE
01    GET PLAYBACK PARAMETERS
01    GETMAIN PLAYBACK BUFFER SPACE
01    MOVE PLAYBACK DCB TO WORK AREA
01    OPEN PLAYBACK DCB INPUT
01    IF PLAYBACK DCB NOT OPENED THEN
02      IF ROUTINE OFFLINE JOB THEN
03        ISSUE ABEND DUMP
02      ELSE
03        ISSUE  MESSAGE 51
02      ENDIF
02      SET RETURN-CODE TO 8
01    ELSE
02      DO UNTIL REQUESTED DATA PLAYED BACK
03        GET RECORD FROM RECORDING/PLAYBACK D.S.
03        IF AN INITIALIZATION RECORD THEN
04          GET DATA RECORD FROM DATA RECORDING/PLAYBACK D.S.
04          IF DATA FALLS WITHIN SPECIFIED TIME RANGE THEN
05            MOVE DATA FROM QSAM BUFFER TO PLAYBACK BUFFER
05            IF DATA TO BE PASSED TO USER PROGRAM THEN
06              LINK TO USER PROGRAM
05            ELSE
06              LINK TO  HEX DUMP ROUTINE
05            ENDIF
04          ENDIF
03        ENDIF
02      ENDIF
01    ENDIF
01    FREE PLAYBACK BUFFER SPACE
01    UNLOCK DPPXDPB
    ENDSEGMENT DPPXDPB
```

Figure 3-153 (1 of 2). DPPXDRC

```
      DPPXDRC MAIN SEGMENT * DATA RECORDING RCUTINE *
01      GET INPUT RECCRD PARAMETERS
01      LCCK DPPXDRC
01      MOVE ADDR OF SYNAD ROUTINE TO CATA RECORDING DCB
01      IF ENABLE ALL ID FLAG CN THEN
01      SET REGISTER 15 TC ZERO
01      ESLE
02        IF ALL ID'S NOT DELETED FROM CATA RECORDING TABLE THEN
03          DO UNTIL END OF DATA RECORDING TABLE
04            IF SPECIFIED ID IN CATA RECORDING TABLE THEN
04            SET REGISTER 15 TO ZERO
04            ENDIF
03          ENCDO
02        ENDIF
01      ENDIF
01      IF ENABLE ID FLAG CN THEN
02        IF DATE IN DATA RECORD TABLE DIFFERENT FRCM CURRENT DATE THEN
03          UPCATE TIME IN DATA RECCRDING TABLE
03          MCVE CATE RECORD HEADER TO QSAM CUTPUT BUFFER
03          PERFORM WRITE RCUTINE
02        ENDIF
02        MOVE CATA RECORD HEADER TO QSAM OUTPUT BUFFER
01      ENDIF
01      UNLOCK DPPXCRC
01      EXIT ROUTINE
01      CPPXCRC WRITE(OUTPUT) ROUTINE
02        IF QSAM BUFFER FULL THEN
03          SET BUFFER FULL FLAG
02        ELSE
03          SET BUFFER FLAG TC NOT FULL
02        ENDIF
02        IF CATA WILL FIT ON BUFFER THEN
03          SET HEADER FLAG TO 0
03          MOVE DATA TO CSAM CUTPLT BUFFER
03          IF QSAM BUFFER HAS LESS THAN 50 BYTES(FULL) REMAINING THEN
04            IF BUFFER NCT CCMPLETLY FULL THEN
05              MOVE PAD RECORD TO QSAM OUTPUT BUFFER(ID=6)
05              ZERO CATA AREA CF PAD RECORD
04            ENDIF
04            WRITE QSAM OUTPUT BUFFER TO CATA RECORDING/PLAYBACK D.S.
03          ENDIF
02        ELSE * IF DATA WILL NCT FIT CN BUFFER THEN *
03          SET HEADER FLAG TO 1
03          MOVE PART CF DATA TO CSAM CUTPUT BUFFER
03          WRITE QSAM OUTPUT BUFFER TO CATA RECORDING/PLAYBACK DATA SET
03          DC UNTIL REMAINING BYTES OF ENTRY ARE MOVED TO CATA SET
```

Figure 3-153 (2 of 2). DPPXDRC

```
04              MOVE SUBHEADER FLAG(2) TO QSAM CUTPUT BUFFER
04              MOVE PART OF DATA TO QSAM OUTPUT BUFFER
04              IF BUFFER HAS LASS THAN 50 BYTES(FULL) REMAINING THEN
05                IF BUFFER NOT COMPLETLY FULL THEN
06                  MOVE PAD RECORD TO QSAM OUTPUT BUFFER(ID=6)
06                  ZERO DATA AREA OF PAC RECORD
05                ENDIF
05                WRITE QSAM OUTPUT BUFFER TO CATA RECORDING/PLAYBACK C.S.
04              ENDIF
03            ENDDO
02          ENDIF
01        ENC DPPXDRC WRITE RCUTINE
        DPPXDRC STAE ROUTINE
01        IF I/O ERROR,PROGRAM CHECK OR ANY ERROR IN TASK DPPXRINT THEN
02          ISSUE ERROR  MESSAGE
02          DISABLE  DATA RECORDING
01        ENDIF
        END CPPXCRC STAE ROUTINE
        ENDSEGMENT CPPXCRC
```

Figure 3-154. DPPXDRCX

```
    CPPXDRCX MAIN SEGMENT  * DUMMY DATA RECORDING ROUTINE *
01    SET RETURN CCDE TO 00
    ENDSEGMENT CPPXDRCX
```

Figure 3-155. DPPXIMPP

```
    DPPXIMPP MAIN SEGMENT * INPUT MESSAGE PROCESSING ROUTINE *
01    BUILD TRANSLATE AND TEST TABLE
01    RETRIEVE PATCH PARAMETERS
01    MOVE INPUT MESSAGE PROCESSING PARAMETERS TC WORK AREA
01    USE GETARRAY MACRO TO FIND INPUT MESSAGE PROCESSING TABLE IN D.B.
01    MOVE IMP CODE IN PARAMETER LIST TO WORK AREA
01    STRTSRCH UNTIL IMP CODE FCUND IN IMP TABLE
02      INCREMENT IMP TABLE TO NEXT ENTRY
01    IF IMP CCDE FOUND IN IMP TABLE THEN
02      IF IMP PARAMETERS WERE PASSED TO DPPXIMPP
03        BUILD PATCH PROBL TO CCNTAIN SPECIFIED IMP PARAMETERS
03        CO UNTIL ALL PARAMETERS ARE PROCESSED
04          IF HEXADECIMAL DATA THEN
05            CCNVERT DATA TO HEXADECIMAL FORMAT
04          ELSE
05            IF FULLWORD DATA THEN
06              CCNVERT DATA TO FULLWORD FCRMAT
05            ELSE   * IF CHARACTER DATA *
06              CCNVERT DATA TO EBIDIC FCRMAT
05            ENDIF
04          ENDIF
03        ENDDO
02      ENDIF
02      PATCH PROCESSING PROGRAM POINTEC TO BY THE IMP CODE IN IMP TABLE
01    ENDIF
01    IF IMP CCDE NCT FOUND IN IMP TABLE THEN
02      ISSUE ERROR MESSAGE
01    ENDIF
01    ENDSRCH
    ENDSEGMENT DPPXIMPP
```

Figure 3-156. DPPXIMPW

```
       DPPXIMPW MAIN SEGMENT * INPUT MESSAGE WTOR ROUTINE *
01       DO WHILE SYSTEM RUNNING
02         ISSUE WTOR TO SYSTEM/370 OPERATOR
02         WAIT FOR SYSTEM/370 OPERATOR REPLY
02         IF STOP COMMAND SPECIFIED THEN
03           TERMINATE JOB STEP WITH ABEND 122
02         ENDIF
02         CALCULATE LENGTH OF OPERATOR REPLY
02         ISSUE WTO MESSAGE # INPUT MESSAGE PROCESSING COMMAND ACCEPTED 4
02         PATCH DPPXIMPP WITH 370 OPERATOR REPLY AS A PARAMETER
01       ENDDO
       ENDSEGMENT DPPXIMPW
```

Figure 3-157. DPPXKILL

```
       DPPXKILL MAIN SEGMENT * CANCEL DUMP ROUTINE *
01       RETRIEVE PASSED PARAMETERS
01       IF OPERATOR COMMENTS PASSED BY IMP COMMAND THEN
02         ISSUE MESSAGE 60 WITH OPERATOR COMMENTS
01       ENDIF
01       IF DUMP PARAMETER PASSED THEN
02         CANCEL THE SPECIAL REAL TIME OPERATING SYSTEM WITH ABEND 122
01       ELSE
02         IF NODUMP PARAMETER PASSED THEN
03           CANCEL THE SPECIAL REAL TIME OPERATING SYSTEM WITH ABEND 222
02         ENDIF
01       ENDIF
       ENDSEGMENT DPPXKILL
```

Figure 3-158. DPPXLOCK

```
     DPPXLOCK - LOCK SUBROUTINE MAIN SEGMENT
01     IF TYPE=LOCK REQUEST
02       IF RESOURCE NOT AVAILABLE
03         CONSTRUCT WAIT CONTROL BLOCK
03         CHAIN WCB ONTO LOCK CONTROL BLOCK'S CHAIN OF WCB'S
03         WAIT ON WCB'S ECB
03         SAVE CURRENT TCB ADDRESS IN LOCK CONTROL BLOCK
02       ENDIF
01     ELSE
02       IF   SOMEONE ELSE WANTS THIS RESOURCE
03           POST THE WCB'S ECB
02       ENDIF
01     ENDIF
     ENDSEGMENT
```

Figure 3-159. DPPXNRTI

```
    DPPXNRTI MAIN SEGMENT * DATA PLAYBACK NON-REAL TIME INITIALIZATICN
01                                ROUTINE *
      LCCK DPPXNRTI
      GET INPUT PARAMETERS
      ZERC OUTPUT WCRK AREA
      IF START DATE = ALL THEN
01      MOVE ALL TC WORK AREA
01      IF LOAD MODULE NAME SPECIFIED THEN
02        MOVE LOAD MOCULE NAME TO WORK AREA
01      ENDIF
      ELSE
01      IF START CATE IS NCN-ZERC THEN
02        MOVE START CATE TO WORK AREA
01      ENDIF
01      MOVE START TIME TO WORK AREA
01      IF STOP CATE IS NON-ZERC THEN
02        MCVE STOP CATE TC WORK AREA
01      ENDIF
01      MCVE STOP TIME TO WORK AREA
01      IF LCAD MOCULE NAME SPECIFIED THEN
02        MOVE LOAC MODULE NAME TC WORK AREA
01      ENDIF
01      DC UNTIL EACH ID HAS BEEN PLACED IN WORK AREA
02        MOVE ID TC WORK AREA
01      ENDDC
      ENDIF
      LINK TO DPPXCCN
      UNLOCK DPPXNRTI
    ENDSEGMENT DPPXNRTI
```

Figure 3-160. DPPXPCON

```
    DPPXPCCN MAIN SEGMENT * CATA PLAYBACK CONVERSICN ROUTINE *
01    GET INPUT PARAMETERS
01    ZERC WCRK AREA
01    MOVE START CATE TO WORK AREA
01    IF START CATE = ALL THEN
02      IF LOAC MCDULE NAME SPECIFIEC THEN
03        MOVE LOAD MODULE NAME TC WORK AREA
02      ENDIF
02      IF OFFLINE JOB THEN
03        SET OFFLINE FLAG IN WORK AREA
02      ENDIF
01    ELSE
02      CCNVERT START TIME FROM EBICIC TO BINARY INTO WORK AREA
02      MCVE STOP CATE TO WORK AREA
02      CCNVERT STOP TIME FRCM EBIDIC TO BINARY INTO WORK AREA
02      MCVE LOAC MODULE NAME TO WORK SPACE
02      CCNVERT IC CCUNT FROM EBIDIC TO BINARY INTO WORK AREA
02      IF OFFLINE JOB THEN
03        SET CFFLINE FLAG IN WORK AREA
02      ENDIF
02      DC UNTIL ALL IC'S ARE CCNVERTEC TO BINARY HALFWORDS
03        CONVERT ID FRCM EBIDIC TO BINARY HALFWORD INTO WORK AREA
02      ENDDC
01    ENDIF
01    LINK TC CPPXDPB *CATA PLAYBACK ROUTINE *
    ENDSEGMENT DPPXPCCN
```

Figure 3-161. DPPXRDR

```
    DPPXRDR MAIN SEGMENT * HEX DUMP ROUTINE *
01    GET ADDRESS OF CATA TO DUMP
01    MCVE PRINT CCB TC WORK AREA
01    OPEN PRINT CCB
01    IF CCB NCT CPENED THEN
02      ISSUE ABENC DUMP
01    ENDIF
01    DO UNTIL INPUT CATA HAS BEEN CUMPED
02      IF BEGINNING CF PAGE THEN
03        PRINT HEADER
02      ENDIF
02      CCNSTRUCT HEX DUMP PRINT LINE
02      PRINT CONSTRUCTED DUMP LINE
01    ENCCO
    ENDSEGMENT CPPXRCR
```

Figure 3-162. DPPXRINT

```
      DPPXRINT MAIN SEGMENT * DATA RECORDING INITIALIZATION ROUTINE *
01      GETINPUT PARAMETERS
01      BUILD LOCK CONTROL BLOCK
01      LOCK DPPXRINT
01      IF DISABLE PARAMETER SPECIFIED THEN
02        IF DATA RECORDING PREVIOUSLY ENABLED THEN
03          PLACE DPPXRINT IN KEY-0
03          ZERO DATA RECORDING TABLE FIELD(SCVTRWA) IN SCVT
03          LOAD DUMMY DATA RECORDING ROUTINE(DPPXDRCX)
03          STORE ADDRESS OF DPPXDRCX IN SCVT
03          PLACE DPPXRINT IN ORIGINAL PROBLEM PROGRAM KEY
03          CLOSE DATA RECORDING DCB
03          FREE QSAM OUTPUT BUFFERS
02        ENDIF
01      ELSE * IF ENABLE PARAMETER SPECIFIED THEN *
02        IF INITIAL DATA RECORDING INITIALIZATION THEN
03          GETMAIN SPACE FOR DATA RECORDING TABLE AND DCB
03          MOVE DATA RECORDING DCB TO GETMAIN AREA
03          OPEN DATA RECORDING DCB
03          IF DCB NOT OPENED THEN
03          ISSUE ERROR MESSAGE
03          ENDIF
03          PLACE DPPXRINT IN KEY-0
03          LOAD DATA RECORDING ROUTINE(DPPXDRC)
03          STORE ADDRESS OF DPPXDRC IN SCVT
03          STORE ADDRESS OF XCVT IN DPPXDRC CONSTANT AREA
03          STORE ADDRESS OF SCVT IN DPPXDRC CONSTANT AREA
03          STORE ADDRESS OF DATA RECORDING TABLE IN SCVT(SCVTRWA)
03          STORE ADDRESS OF DATA RECORDING TABLE IN DPPXDRC CONSTANT AREA
03          STORE ADDRESS OF TIME ARRAY IN DPPXDRC CONSTANT AREA
03          PLACE DPPXRINT IN ORIGINAL PROBLEM PROGRAM KEY
03          MOVE LOCK CONTROL BLOCK NAME TO DPPXDRC CONSTANT AREA
03          STORE ADDRESS OF LOCK CONTROL BLOCK ADDRESS IN DPPXDRC
03          RETRIEVE A QSAM OUTPUT BUFFER
03          STORE ADDRESS OF OUTPUT BUFFER IN DPPXDRC CONSTANT AREA
03          MOVE COPY OF TIME ARRAY TO DATA RECORDING TABLE
02        ENDIF
02        IF ADD PARAMETER SPECIFIED THEN
03          STORE PASSED ID PARAMETERS IN DATA RECORDING TABLE
02        ELSE
03          IF DELETE PARAMETER SPECIFIED THEN
04            DELETE PASSED ID'S FROM DATA RECORDING TABLE
03          ENDIF
02        ENDIF
02        IF ALL PARAMETER SPECIFIED THEN
03          SET DATA RECORDING ID COUNT TO ENABLE ALL(255)
02        ENDIF
02        ENDIF
02        SET DATA RECORDING ID COUNT IN DATA RECORDING TABLE
01      ENDIF
      ENDSEGMENT DPPXRINT
```

Figure 3-163. DPPXRPRT

```
      DPPXRPRT MAIN SEGMENT * REPCRT CATA OUTPUT FACILITY *
01      MOVE NAME OF PASSED OUTPUT DDNAME TO DCB
01      OPEN OUTPUT CATA SET DCB
01      IF OUTPUT CATA SET DCB NOT CPENED THEN
02        ISSUE ERROR MESSAGE * SYSTEM MESSAGE 53 *
01      ELSE
02        IF NEW PARAMETER PASSED THEN
03          IF OUTPUT CATA SET ON TAPE THEN
04            REWIND TAPE DRIVE TO BEGINNING OF DATA SET
03          ELSE
04            IF OUTPUT DATA SET CN DISK THEN
05              POSITICN DISK DRIVE AT BEGINNING OF DATA SET
04            ENDIF
03          ENDIF
02        ENDIF
02        DO UNTIL ALL INPUT DATA SETS MOVED TO OUTPUT DATA SET
03          MOVE NAME CF PASSED INPUT DDNAME TO DCB
03          OPEN INPUT DATA SET DCB
03          IF INPUT CATA SET CCB NCT OPENED THEN
04            ISSUE ERROR MESSAGE * SYSTEM MESSAGE 53 *
03          ELSE
04            DC UNTIL ENTIRE INPUT CATA SET MCVED TO OUTPUT DATA SET
05              MOVE DATA RECORD FROM INPUT CATA SET TO OUTPUT DATA SET
04            ENDDO
03          ENDIF
03          CLOSE INPUT DATA SET DCB
02        ENDDO
01      ENDIF
01      CLCSE OUTPOT DATA SET DCB
      ENDSEGMENT DPPXRPRT
```

Figure 3-164. DPPXSVCP

```
   DPPXSVCP-MAIN SEGMENT
    INITIALIZE WORK AREAS
    IF INPUT APRAMETERS ARE VALID
01      BUILD PARAMETER TO RETURN TO INITIAL PSW STATUS
01      CALL  MODESET TO CHANGE PSW
    ENDIF
    LOAD ADDRESS OF TABLE OF NUCLEUS ADDRESSES
   ENDSEGMENT
```

Figure 3-165 (1 of 10). DPPXUTIL

```
   MAIN SEGMENT OF DPPXUCTL
   SAVE REGS
   GET SAVE/WORK AREA
   CLEAR GETMAIN AREA
   OPEN PRINT CCB
   IF NO PARM FLD CN EXEC CARD,THEN
01   SET ASM OPTICN TO 'F'
   ELSE
01   VALIDITY CHECK PARM AND SET ASM OPTION
01   NOPDL
01   IF PARM=H,OR
01   IF PARM='H,NOGEN',THEN
01   ELSE
02     IF PARM='H,GEN',THEN
02     ELSE
03       IF PARM='F',OR
03       IF PARM='NCGEN',OR
03       IF PARM='F,NOGEN',THEN
03       ELSE
04         IF PARM='GEN',OR
04         IF PARM='F,GEN',THEN
04         ELSE
04         ENDIF
03       ENDIF
02     ENDIF
01   ENDIF
   ENDIF
   IF ASM CPTICN IS H,THEN
01   BUILD 'H' ASSEMBLER PARM LIST
   ELSE
01   BUILD 'F' ASSEMBLER PARM LIST
   ENDIF
   SET UP LCADER PARM LIST
   SET UP IEBUPDTE PARM LIST
   SAVE SVC FOR ABEND PROCESSING
   OPEN SYSPRINT ANC SYSIN
   IF SYSPRNT NT OP,THEN
01   SET RETURN CODE TO 16
01   TAKE ERROR EXIT-ALL
   ENDIF OPEN PRINTDCB
   PRINT OFFLINE UTILITY HEADER LINE
   IF ERR IN EXEC CRD PRM FLD,THEN
01   PRINT PARM FIELD ERROR MESSAGE
   ENDIF
   IF SYSIN NCT CPEN
01   PRINT MISSING DD-CARD MESSAGE
01   SET RETURN CODE TC 12
01   TAKE ERROR EXIT-ALL
   ENDIF CPEN INCCB
```

Figure 3-165 (2 of 10). DPPXUTIL

```
     UNTIL END OF FILE ARISES FRCM SYSIN,DO
01     IF CCNTROL CARD NCT SAVED,THEN
02       GET ONE CARD FRCM SYSIN
01     ENDIF NOT SAVED
01     PRINT CARD IMAGE TO SYSPRINT
01     IF CARD IS NOT CONTROL CARD
02       PRINT FIRST CARD MUST BE A CONTROL CARD MSG
01     ELSE
02       IF CCNTFCL CARD VALID
03         PRINT CONTROL CARD VALIC MESSAGE
03         OPEN SYSUT4 DD CARD FOR OUTPUT
03         IF OPEN NCT SUCC.
04           PRINT MISSING DD-CARD MSG
04           SET RETURN CODE TO 12
04           TAKE ERRCR EXIT-ALL
03         ENDIF OPEN UTDCB
03         IF ',TFEN IF CNLINE BUILC CTRL CARD
04           IF NOGEN ASM OPTION SELECTEC,TFEN
05             PUT PRINT NOGEN CARD TO SYSUT4
04           ENDIF
04           BUILD AREA CEFINITION MACRO FROM CONTROL CARD
04           PUT AREA DEFINITION MACRC TO SYSUT4
04           DO DPPXUTL2 - INPUT PROCESS CCNTROL
04           BAD - ENTER HERE FROM BAD INPUT DDNME EXIT
03         ELSE
04           DO DPPXUTL4 - IEBUPDTE INTERFACE ROUTINE
03         ENDIF DPPXUCTL
02       ELSE IF CONTRCL CARD IS INVALID
03         PRINT INVALID, SKIPNG FOR NXT CTL CRD MSG
03         SET EODAD RETURN ADDR - NCEOFIN
03         UNTIL NEXT CNTL CARD,CO
04           FLUSH CATA CARDS
03         ENDDO FLLSH
02       ENDIF SWITCFES
01     ENDIF #/
01     NOWODIN - ENTER HERE FROM END-OF-CATA RCUTINE
     ENDDO ENDOFILE
     PRINT END CF UTILITY DPPXUTIL MSG
     CLOSE SYSPRINT AND SYSIN DCB'S
     ALL - ENTER HERE FRCM ERROR EXIT
     SAVE RETURN CODE
     IF DTA BSE OPERATION WAS DONE,TFEN
     ENDIF
     FREE SAVE/WORK AREA
     RESTORE RETURN CODE
     RESTCRE REGS 0 - 12
     RETURN
     BGNSEG CCNTFOL CARD VALIDITY CFECK ROUTINE
```

Figure 3-165 (3 of 10). DPPXUTIL

```
01   FIND NONBLANK CHAR WITHIN COL 3-63
01   IF NCNBLANK CHAR FOUND,THEN
02     IF ',OR   IF CNLINE BUILD CCNTROL CARC,OR
02     IF ',THEN IF UPCATE CCNTROL CARD,THEN
03       SAVE OPERATION
03       IF ANY OPERAND FOUND,THEN
04         UNTIL BYPASS BIT IS SET,OR
04         UNTIL '),DO  UNTIL BLANK IS FOUND,DO
05           IF THIS CHAR IS CCMMA,THEN
06             INCREMENT BY 1
05           ENDIF CCMMA
05           IF ',OR  IF THIS CHAR IS BLANK,CR
05           GET ADDR CF CCNT MARK
05           IF THIS IS COLUMN 72,THEN
06             IF ',THEN IF NO CONTMARK,THEN
07               PRINT PARM OR CONTMARK MISSING MSG
07               SET BYPASS SWITCH
06             ELSE
07               GET CCNTINUATION CARD
07               PRINT CARD IMAGE TO SYSPRINT
07               IF ',THEN IF COL 1-15 ARE BLNK,THEN
08                 IF ',THEN IF COL 16 IS BLANK,THEN
09                   PRINT EXPECTED CONT NOT RECVD MSG
09                   SET BYPASS SWITCH
08                 ENDIF COL16 BLANK
07               ELSE
08                 PRINT COL 1-15 MUST BE BLANK MSG
08                 SET BYPASS SWITCH
07               ENDIF COL 1-15 BLANK
06             ENDIF CONTMARK
05           ENDIF NEXT CHAR BLANK
05           IF BYPASS SWITCH IS NCT SET,THEN
06             FIND NEXT DELIMITER
06             IF DELIMITR FOUND,THEN
07               CALCULATE LENGTH OF ENTITY
06             ELSE
07               IF ',THEN IF COL72 IS BLANK,THEN
08                 CALCULATE LENGTH OF ENTITY
07               ELSE
08                 PRINT CTL CARD TEXT BEYOND 71 MSG
08                 SET BYPASS SWITCH
07               ENDIF BLANK
06             ENDIF DELIMITR
05           ENDIF BYPASS1
05           IF BYPASS SWITCH IS NOT SET,THEN
06             IF ',THEN IF ONLINE BUILC CNTL CRD
07               IF THIS IS AREA KEYWRD
```

Figure 3-165 (4 of 10). DPPXUTIL

```
08          IF FIRST TIME AREA KEYWORD CNTL CRD
09            CALC ADDRESS OF PARAMETER
09            CALC LENGTH OF PARAMETER
09            IF AREA LEN IS NOT ZERO,THEN
10              IF AREA=DISPDEF,OR
10              IF AREA=MSGDEF,OR
10              IF AREA=DBDEF,THEN
11                SAVE AREA NAME
11                IF AREA,NE,DBDEF,AND    DR#5451
11                IF DB OPN WAS DONE,THEN    DR#5451
12                  TURN OFF DATA BASE FLAG      DR#5451
12                  LINK TO DPPXDBIN             DR#5451
11                ENDIF DR#5451
10              ELSE
11                DO WPARM - WRONG PARAMETER ROUTINE
10              ENDIF AREA PARAMETERS
09            ENDIF AREA NULL
08          ELSE
09            DO NOTFST - NOT FIRST TIME FOR KEYWORD ROUTINE
08          ENDIF AREAL
07        ELSE
08          IF INPUT KEYWRD,THEN
09            IF FIRST INPT KEYWRD ON CNTL CRD
10              CALC ADDRESS OF PARAMETER
10              CALC LENGTH OF PARAMETER
10              IF INPUT LEN IS NOT ZERO,THEN
11                IF NOT INPUT=*,THEN
12                  CALC LENGTH OF DDNAME
12                  DO NAMCHK CHK VALIDTY DDNME
12                  IF DELIMITER WAS (,THEN
13                    GET ADDRESS OF MEMBER NAME
13                    GET ADDRESS OF LAST CHAR
13                    CALC LENGTH OF MEMBER
13                    IF NO RIGHT PAREN
14                      ADJUST LENGTH
14                      PRINT RIGHT PAREN MISSING MSG
13                    ENDIF RIGHT PAREN
13                    DO NAMCHK CHK VALIDTY MEM
12                  ENDIF LEFT PAREN
11                ELSE
12                  STORE DDNAME LENGTH
12                  SAVE ASTERISK
11                ENDIF INPUT EQ *
10              ENDIF INPUT NULL
09            ELSE
10              DO NOTFST - NOT FIRST TIME FOR KEYWORD ROUTINE
09            ENDIF INPUTL
```

Figure 3-165 (5 of 10). DPPXUTIL

```
08                  ELSE
09                    IF OPTN KEYWD,THEN
10                      IF FIRST TIME
11                        CALC ADCRESS
11                        CALC LENGTH
11                        IF OPTION LEN IS NOT ZERO,TFEN
12                          IF CPTICN=ADD,OR
12                          IF OPTICN=DEL,OR
12                          IF CPTICN=REPL,OR
12                          IF OPTICN=TEST,THEN
13                            SAVE PARAMETER
12                          ELSE
13                            DO WPARM - WRCNG PARAMETER ROUTINE
12                          ENDIF OPTION PARAMETERS
11                        ENDIF OPTION NULL
10                      ELSE
11                        DO NOTFST - NOT FIRST TIME FOR KEYWORD RCUTINE
10                      ENDIF OPTIONL
09                    ELSE
10                      IF TYPE KEYWD,THEN
11                        IF FIRST TIME
12                          CALC ADCRESS
12                          CALC LENGTH
12                          IF TYPE LEN IS NOT ZERO,THEN
13                            IF TYPE PARM LT 7 BYTES
14                              SAVE PARAMETER
13                            ELSE
14                              PRINT PARM IN ERROR MSG
14                              SET INVALIC SWITCH
13                            ENDIF LENGTH LESS 7
12                          ENDIF TYPE NULL
11                        ELSE
12                          DO NOTFST - NOT FIRST TIME FOR KEYWORD ROUTINE
11                        ENDIF TYPEL
10                      ELSE
11                        DO WKEYW - WRONG KEYWORD ROUTINE
10                      ENDIF TYPE
09                    ENDIF OPTION
08                  ENDIF INPUT
07                ENDIF AREA
06              ELSE IF UPCATE OPERATION
07                IF NEWSET KEYWD,THEN
08                  IF FST NEWSET KEYWD ON CNTL CD
09                    CALC ADDRESS
09                    CALC LENGTH
09                    DC NAMCHK - CHK VALIDTY NEWSET
08                  ELSE
09                    DC NOTFST - NOT FIRST TIME FOR KEYWCRD ROUTINE
08                  ENDIF NEWSETL
```

Figure 3-165 (6 of 10). DPPXUTIL

```
07               ELSE
08                 IF CLDST KEYWD,THEN
09                   IF FIRST TIME
10                     CALC ADDRESS
10                     CALC LENGTH
10                     DC NAMCHK - CHK VALIDTY OLDST
09                   ELSE
10                     DO NOTFST - NOT FIRST TIME FOR KEYWORD ROUTINE
09                   ENDIF OLDSETL
08                 ELSE
09                   DO WKEYW - WRONG KEYWORD ROUTINE
08                 ENDIF OLDSET
07               ENDIF NEWSET
06             ENDIF DPPXUCTL
05           ENDIF BYPASS2
04         ENDDO BYPASS OR BLANK
04         IF NC SWITCH SET
05           IF ',THEN IF ONLINE BLD OPERATN,THEN
06             IF NC INPUT WAS SPECIFIED,THEN
07               PRINT INPUT SPEC MISSING MSG
07               SET INVALID SWITCH
06             ENDIF INPUTL
06             IF NO AREA KEYWORD WAS SPECFD,THEN
07               PRINT AREA SPEC MISSING MSG
07               SET INVALID SWITCH
06             ENDIF AREAL
05           ELSE IF UPDATE OPERATICN,THEN
06             IF NC NEWSET WAS SPECIFIEC,THEN
07               PRINT NEWSET SPEC MISSING MSG
07               SET INVALID SWITCH
06             ENDIF NEWSETL
06             IF NC OLDSET WAS SPECIFIED,THEN
07               PRINT OLDSET SPEC MISSING MSG
07               SET INVALID SWITCH
06             ENDIF CLDSETL
05           ENCIF DPPXUCTL
04         ENDIF SWITCHES
03       ELSE IF NO OPERAND FOUND, ALL BLANKS
04         PRINT NO CPERAND FOUND MSG
04         SET BYPASS SWITCH
03       ENDIF OPERAND
02     ELSE IF NO VALID OPERATICN FCUND
03       PRINT INVALID OPERATION MSG
03       SET BYPASS SWITCH
02     ENDIF VALIDOP
01   ELSE IF NC OPERATION FOUND, COL 3-63 ALL BLANKS
02     PRINT NC CPERATICN FCUND MSG
02     SET BYPASS SWITCH
01   ENDIF OPERATION
01   MARK CCNTFOL CARD AS PROCESSED
```

Figure 3-165 (7 of 10). DPPXUTIL

```
    ENDSEG DPPXUTL1
    END OF VALIDITY CHECK ROUTINE
    BGNSEG DPPXUTL2 - INPUT PROCESSING CONTROL
01    IF INPUT FROM SYSIN,THEN
02      GET A(DCB) FOR READ FROM SYSIN
02      DO DPPXUTL3 - INPUT READ/WRITE
01    ELSE
02      IF INPUT FROM SEQ DATA SET,THEN
03        SET UP AND GET A(DCB) FOR SEQ INPT READ
03        OPEN REMOTE SEQ INPUT DCB
03        IF NO DD
04          PRINT MISSING DD CARD MSG
04          CLOSE SYSUT4 DCB
04          FREEPOOL SYSUT4
04          TAKE BAD INPUT EXIT - BAD
03        ENDIF
03        GET A(DCB) FOR READ FROM SEQ INPUT
03        DO DPPXUTL3 - INPUT READ/WRITE
03        CLOSE SEQ INPUT DCB
02      ELSE
03        SET UP PDS INPUT DCB
03        GET A(DCB) FOR READ
03        OPEN REMOTE PDS INPUT DCB
03        IF NO DD CRD
04          PRINT MISSING DD CARD MSG
04          CLOSE SYSUT4
04          FREEPOOL SYSUT4
04          TAKE BAD INPUT DDNAME EXIT - BAD
03        ENDIF
03        DO DPPXUTL3 - INPUT READ/WRITE
03        CLOSE PDS INPUT DCB
02      ENDIF
01    ENDIF
01    DO DPPXUTL5 - ASM/LOAD/FPP INTERFACE
    ENDSEG DPPXUTL2
    END OF INPUT PROCESSING CONTROL ROUTINE
    BGNSEG *:DPPXUTL3 - INPUT READ/WRITE
01    SET UP EODAD RETURN ADDRESS-ENDSEG3
01    WHILE INPUT NE CNTL CRD,DO
02      GET CARD FROM INPUT DATA SET
02      IF CARD NE CONTROL CRD,THEN
03        PUT CARD TO SYSUT4
02      ENDIF
01    ENDDO
01    ENDSEG3 - ENTRY FOR END-OF-DATA
01    IF UPDATE OPERATION,THEN
02      IF AREA=DBDEF,THEN
03        PUT DBEND MACRO CALL IN OUTPUT
```

Figure 3-165 (8 of 10). DPPXUTIL

```
02      ELSE
03        IF AREA=MSGDEF
04          PUT MSGEND MACRO CALL IN OUTPUT
03        ELSE
04          PUT DISPEND MACRO CALL IN OUTPUT
03        ENDIF
02      ENDIF
02      PUT END MACRC TO SYSUT4
01    ENDIF
01    CLOSE SYSUT4
01    FREEPCOL SYSUT4
    ENDSEG DPPXUTL3
    END OF INPUT READ/WRITE SUBROUTINE
    BGNSEG CPPXUTL4 - IEBUPDTE INTERFCE
01    DO DPPXUTL3 - INPUT READ WRITE
01    MOVE OLDSET DCNAME TC PARM LIST
01    MOVE NEWSET DDNAME TO PARM LIST
01    LINK TO IEBUPDTE
    ENDSEG CPPXUTL4
    END OF IEBUPDTE INTERFACE ROUTINE
    BGNSEG DPPXUTL5 - ASM/LOADER/FPP INTERFACE
01    IF ASM 'H' SPECIFIED,THEN
02      LINK TO IEV90
01    ELSE
02      LINK TO IFOX00
01    ENDIF
01    IF DR#2056
01    ENDIF DR#2056
01    SAVE RETURN CODE
01    IF RETURN CD IS LT 8,THEN
02      LCAD HEWLOADR
02      CALL HEWLCADR
02      IF RETURN CD IS LT 8,THEN
03        SAVE RETURN CODE
03        LINK TO FINAL PHASE PROCESSOR
03        SAVE RETURN CCDE
02      ELSE
03        SAVE RETURN CCDE
03        PRINT BAD RETURN CODE FRCM LOADER MSG
02      ENDIF
02      IF LOADR STORAGE NOT FREEC BY FPP
03        IF * IF LENGTH IS NOT ZERO
04          FREE LOADER GOTTEN STORAGE
03        ENDIF
02      ENDIF
01    ELSE
02      PRINT BAD RETURN CODE FRCM ASSEMBLER MSG
01    ENDIF
```

Figure 3-165 (9 of 10). DPPXUTIL

```
   ENDSEG DPPXUTL5
   END OF    ASM/LOAD/FPP INTERFACE
   BGNSEG STAE EXIT ROUTINE
01   SAVE REGS
01   GET A(FIRST SAVE AREA)
01   GET A(DPPXUTIL WORKAREA)
01   IF DTA BSE CPERTN WAS DONE,THEN
02     LINK TO DPPXCBIN TO BUILC ONLINE TABLES
01   ENDIF
01   SET NO RETRY
01   RESTORE REGS 0-12
   ENDSEG STAEEXIT             STAE EXIT ROUTINE
   BGNSEG PRINT ROUTINE
01   FORMAT MESSAGE
01   PUT A LINE TO SYSPRINT
   ENDSEG PRINT                  PRINT ROUTINE
   BGNSEG WRCNG PARAMETER ROUTINE
01   IF PARM LEN NE ZERO,THEN
02     MOVE ENTITY TO OUTPUT LINE
01   ENDIF
01   PRINT WRONG PARAMETER MSG
01   SET INVALID SWITCH
   ENDSEG WPARM                  WRONG PARAMETER ROUTINE
   BGNSEG NOT FIRST TIME FOR KEYWORD RCUTINE
01   IF KEYWORD LEN NE ZERO,THEN
02     MOVE KEYWORD TC OUTPUT LINE
01   ENDIF
01   PRINT MULTIPLE KEYWORD MSG
01   SET INVALID SWITCH
   ENDSEG NOTFST                 NCT FIRST TIME FOR KEYWORD ROUTINE
   BGNSEG WRONG KEYWORD ROUTINE
01   IF KEYWORD LENGTH NE ZERO,THEN
02     MOVE KEYWORD TO OUTPUT LINE
01   ENDIF
01   PRINT WRONG KEYWORD MSG
01   SET INVALID SWITCH
   ENDSEG WKEYW                  WRONG KEYWORD RCUTINE
   BGNSEG NAME VALIDITY CHECK ROUTINE
01   IF NAME LEN NE ZERO,THEN
02     IF LENGTH GT 8,CR
02     IF FIRST CHAR NOT ALPHA,OR
02     IF CHAR NOT WITHIN X'CO' - X'FF',OR
02     IF CHAR NOT ALPHANUMERIC,THEN
03       PRINT PARAMETER IN ERROR MSG
03       SET INVALID SWITCH
```

Figure 3-165 (10 of 10). DPPXUTIL

```
02      ELSE
03        STORE LENGTH
03        SAVE NAME
02      ENDIF ALPHA
01    ENDIF PREVENT
   ENDSEG NAMCHK              NAME VALIDITY CHECK ROUTINE
   BGNSEG SYSIN END-OF-DATA ROUTINE
01    PRINT EOF ON SYSIN MSG
01    SET EOF ON SYSIN INDICATOR
   ENDSEG EODAD   SYSIN END-OF-DATA ROUTINE
   BGNSEG INPUT END-OF-DATA ROUTINE
01    PRINT SEQ DATA SET EOF MSG
   ENDSEG SEQEODAD INPUT END-OF-DATA ROUTINE
```

Figure 3-166. DPPZSAMP

```
       DPPZSAMP MAIN SEGMENT 'SAMPLE PROGRAM'
01       PATCH DPPSAMP1
01       IF PATCH SUCCESSFUL THEN
02         ISSUE SYSTEM MESSAGE 68
01       ENDIF
01       GET ARRAY DPPZSAMP FROM DATA BASE
01       IF ARRAY RETRIEVED THEN
02         DISPLAY CONTENTS OF ARRAY VIA MESSAGE 66
02         LOG OUT ARRAY DPPZSAMP
02         IF ARRAY LOGGED THEN
03           ISSUE SYSTEM MESSAGE 68
03           LOG IN ARRAY DPPZSAMP
03           IF ARRAY LOGGED IN THEN
04             DISPLAY CONTENTS OF ARRAY VIA MESSAGE 66
04             PLACE LOGGED IN ARRAY IN DATA BASE
04             IF LOGGED IN ARRAY PLACE IN DATA BASE THEN
05               ISSUE SYSTEM MESSAGE 68
04             ENDIF
03           ENDIF
02         ENDIF
01       ENDIF
01       GET ITEM DPPSAMP2 FROM DATA BASE
01       IF ITEM RETRIEVED THEN
02         ISSUE SYSTEM MESSAGE 69
02         PLACE ITEM IN DATA BASE
02         IF ITEM DPPSAMP2 PLACED IN DATA BASE THEN
03           ISSUE SYSTEM MESSAGE 68
02         ENDIF
01       ENDIF
01       ISSUE PTIME MACRO TO CAUSE TASK DPPSAMP1 TO BE PATCHED 3 TIMES AT
02            1 SECOND INTERVALS
01       IF PTIME MACRO SUCCESSFUL THEN
02         ISSUE SYSTEM MESSAGE 68
01       ENDIF
       ENDSEGMENT DPPZSAMP
```

Figure 3-167 (1 of 2). DPTCSVC1

```
   IF NO INVALID ADDRESS FOUND IN LIST
01   IF THERE ARE ANY BLKS CN CHAIN
02     IF NOT - FALL THRU
03       IF TYPE = ADD
04         IF QPOS = LAST
05           WHILE LOOP TILL LAST BLK IN CHN FOUND
05           BGNWHILE
05           ENDDO
04         ELSE
05           GET ADDR OF CHAINORG FULLWORD
05           CHAIN NEW BLK AT BEGINNING
04         ENDIF
03       ELSE
04         IF FRST BLK= RMV BLK
05           GET ADDR CF CHAINCRG FULLWORD
05           POINT ORG TO BLK AFTER RMV BLK
04         ELSE
05           STRTSRCH LOOP TILL END OF CHAIN FCUND
05           EXITIF EXIT FROM SEARCH IF RMV BLK FND
06             GET PTR BLOCK FCLLOWING ONE TO
06             *    BE REMOVED
06             DECHAIN THE BLOCK
05           ORELSE
06             UPDATE CURRENT BLK PTR
05           BGNWHILE
05           ENDLOOP
06             INDICATE ECB PREVIOUSLY POSTED
05           ENDSRCH
04         ENDIF
03       ENDIF
02     ELSE
03       GET ORDER WORD OFFSET
03       GET ORDER WORD FOR BLK TO ADD
03       SEE IF NEW BLK TO BE FIRST
03       IF ORDER WORD LT ORDER WORD OF NXT BLK
04         POINT ORG AT NEW BLK
04         POINT NEW BLK TO REST OF CHAIN
03       ELSE
04         STRTSRCH LOOP TILL END OF CHAIN
04         EXITIF EXIT IF NEXT GT CURRENT
05           CHAIN NEW BLOCK IN CHAIN
04         ORELSE
05           UPDATE CURRENT BLK PTR
04         BGNWHILE
04         ENDLCOP
05           CHAIN NEW BLK AT END
05           ZERC NEXT PTR IN NEW BLK
04         ENDSRCH
03       ENDIF
02     ENDIF
```

Figure 3-167 (2 of 2). DPTCSVC1

```
01    ELSE
02      IF BLOCK TO BE ADDED
03        CHAIN THE NEW BLK AT BEGINNING
02      ELSE
03        SET COND CODE=4 - BLK NOT FOUND
02      ENDIF
01    ENDIF
01    IF NO ERRORS FOUND
02      GET THE ECB ADDR
02      IF AN ECB IS TO BE POSTED
03        IF THE ECB HAS NOT ALREADY BEEN POSTEDEN
04          GET POST BR ENTRY ADDRESS
04          BRANCH TO POST TO HAVE ECB POSTD
03        ELSE
04          INDICATE ECB PREVIOUSLY POSTED
03        ENDIF
02      ENDIF
01    ENDIF
    ENDIF
```

Figure 3-168. DPTDLMP1

```
    BGNSEG DPTDLMP1 - TCBX-LCB USER SCAN
01    WHILE MORE TCBX'S ARE CHAINED
02      GET TCBX-LCB CHAIN ORIGIN
02      WHILE MORE LCB'S ARE CHAINED
03        UNTIL ALL MODULE NAMES EXAMINED
04          IF NAME EQUAL TO LCBEPNAM
05            SET FLAG LOAD MODULE PURGE REQ
05            IF MODULE IS CURRENTLY EXECUTING
06              SET UP AN ECB FOR DPPTPMON
06              CLEAR THE ECB
06              INCREMENT ECB COUNT
05            ENDIF MODULE IS EXECUTING
05            CAUSE EXIT OF BXLE LOOP
04          ENDIF NAME EQ
03        ENDDO UNTIL BXLE
02      BGNWHILE
03        GET ADDR OF NEXT LCB IN CHAIN
02      ENDDO WHILE LCB
01    BGNWHILE
02      GET ADDR OF NEXT TCBX IN CHAIN
01    ENDDO WHILE TCBX
    ENDSEG DPTDLMP1
```

Figure 3-169. DPTDLMP2

```
   BGNSEG DPTDLMP2 - TCBX-LCB PURGE SCAN
01   WHILE MORE TCBX'S ARE CHAINED
02     GET TCBX-LCB CHAIN ORIGIN
02     TURN OFF INTERNAL DELETE FLAG
02     WHILE MCRE LCB'S ARE CHAINEC
03       IF STIMER HAS NOT EXPIRED
04         IF PURGE REQUESTED
05           IF NOT LOADED BY THIS TASK
06             TURN OFF PURGE REQUEST FLAG
06             SET LCB UNRESOLVED
05           ELSE - IF LOADED BY THIS TASK
06             IF MODULE IN STORAGE
07               SET INTERNAL CELETE FLAG
06             ELSE
07               TURN OFF PURGE FLAG
06             ENDIF
05           ENDIF NCT LOADED BY THIS TASK
04         ENDIF NO PURGE REQ
03       ELSE - IF STIMER HAS EXPIRED
04         TURN OFF PURGE FLAG IN THE LCB
03       ENDIF STIMER EXPIREC
02     BGNWHILE
03       GET ADDR OF NEXT LCB IN CHAIN
02     ENDDO WHILE LCB
02     IF INTERNAL DELETE FLAG SET
03       CREATE AN IRB
03       SET UP AN ECB
03       CLEAR THE ECB
03       INCREMENT ECB COUNT
03       INITIALIZE IQE
03       STORE IRB ADCR INTO IQE
03       MOVE TCB ADDR INTO IQE
03       CALL STAGE 2 EXIT EFFECTOR
02     ENDIF PMONDEL
01   BGNWHILE
02     GET ADDR OF NEXT TCBX IN CHAIN
01   ENDDO WHILE TCBX
   ENDSEG CPTDLMP2
```

Figure 3-173 (2 of 2). DPTDSVC1

```
05           SET TCBXNAME NCT FCUND FLAG
04         ENDSRCH
04         IF PTN=FIND WAS SPECIFIED
05           SWITCH PARTITIONS
04         ENDIF
03       ENDDO UNTIL DSVCLOOP
03       IF TCBXNAME NCT FOUND FLAG SET
04         SET RC FOR TCBXNAME NCT FOUND
03       ENDIF
02     ENDIF RC NZERO
01   ENDIF NO TCBXNAME
   ENDIF RC ZERO
```

Figure 3-174 (1 of 2). DPTPMON1

```
   IF DDS WAS INVOKED
01   EXEC DDS CLEANUP ROUTINE
   ENDIF
   WHILE WQE'S CHAINED TO TCBXCUWQ
01   TOP WQE BECOMES THE CURRENT
01   DECHAIN TOP WQE FROM TCBXCUWQ
01   EXEC WQE-DELETE SVC ROUTINE
   BGNWHILE
01   GET ADDR OF TOP WQE ON TCBXCUWQ
   ENDDO WHILE TCBXCUWQ NZERO
   WHILE WQE'S CHAINED TO TCBXWQ
01   TOP WQE BECOMES THE CURRENT
01   DECHAIN TOP WQE FROM TCBXWQ
01   EXEC WQE-DELETE SVC ROUTINE
   BGNWHILE
01   GET ADDR OF TOP WQE ON TCBXWQ
   ENDDO WHILE TCBXWQ NZERO
   CLEAR CURRENT WQE POINTER
   GET ADDR OF AT-TYPE DUMMY GFBE
   IF DUMMY GFBE DOESN'T POINT AT ITSELF
01   FREEWA ALL AT-TYPE GETWA AREAS
   ENDIF
   GET TCBX-LCB CHAIN ORIGIN
   WHILE MORE LCB'S ARE CHAINED TO THE TCBX
01   IF AN ASSOC LCB IS ON TMCT-LCB CHAIN
02     DECREMENT USECOUNT IN TMCT-LCB
02     IF USECOUNT BECOMES ZERO
03       SET FLAG TO CAUSE DELETE
03       CAUSE DPPTSMON TO SCAN LCB'S FOR DELETE
02     ENDIF USECOUNT ZERO
01   ENDIF LCB ON TMCT-LCB CHAIN
   BGNWHILE
01   GET NEXT LCB ON TCBX-LCB CHAIN
   ENDDO
   IF TCBXNAME BLANK - DEPENDENT TASK
01   GET DEPENDENT TASK CHAIN ORIGIN
   ELSE - INDEPENDENT TASK
01   GET INDEPENDENT TASK CHAIN ORIGIN
   ENDIF
   STRTSRCH WHILE MORE TCBX'S ARE CHAINED
   EXITIF THIS IS OUR TCBX
01   DECHAIN TCBX FROM ITS TCBX-CHAIN
   ORELSE
01   GET NEXT TCBX ON CHAIN
   ENDLOOP - TCBX WAS NOT ON CHAIN
01   ISSUE MESSAGE DPP016 TCBX NOT FOUND
   ENDSRCH
```

Figure 3-174 (2 of 2). DPTPMON1

```
   DECREMENT # CF ACTIVE TCBX'S IN TMCT
   IF TMCT#FRE  GT  DELT#FRE  AND
   IF TMCT#ACT + TMCT#FRE GT TMCT#TCB
01   * THIS TASK HAS TO TERMINATE
01   SET FLAG TO CAUSE TERMINATICN
01   * DPPTETXR WILL FREE TCB AND TCBX
   ELSE
01   * LEAVE TFE TASK ALIVE
01   UPDATE # OF FREE TCBX'S IN TMCT
   ENDIF
   IF THIS TASK HAS A SUBTASK
01   SET FLAG TO CAUSE TERMINATION
   ENDIF
   WHILE MCRE LCB'S ARE CHAINEC TO THE TCBX
01   IF
01   IF MODULE LCACED BY THIS TASK
02     DELETE THE MODULE
01   ENDIF
01   DECHAIN THE LCB
01   CB-FREE THE LCB
   BGNWHILE
01   GET NEXT LCB CN TCBX-LCB CHAIN
   ENDDO
   IF THIS TASK SHALL NOT TERMINATE
01   MAKE TCBX LOCK LIKE UNUSED
01   CHAP THE TASK DOWN TO ZERO PRTY
01   CLEAR RESOURCE TABLE
01   CHAIN TCBX TC FREE CHAIN
   ENDIF
```

Figure 3-175. DPTPMON2

```
     IF NO LOAD MCDULE PURGE REQUESTED AND
     IF NO DELETE REQUESTED
01     UNTIL ANY DPATCH DC
02       WHILE MORE WQE'S CHAINEC AND
02       WHILE NC DPATCH U
03         GET LCB PCINTED TO BY NEXT WQE
03         IF SAME LCB = SAME ROUTINE AS BEFORE
04           DO CPTPMON5 - PERFORM WQE-CLEANUP
04           TOP WQE BECCMES CURRENT
04           DECHAIN CURRENT WQE
04           IF CURRENT QUEUE LENGTH NOT ZERO
05             DECREMENT CURRENT QUEUE LENGTH
04           ENDIF CQL NZERO
04           IF NCT SPECIAL ID
05             IF PROBL WAS MOVED INTO WQE
06               LOAD ADDR OF PROBL IN WQE
05             ELSE
06               GET ADDR OF USER'S PROBL
05             ENDIF
05             STORE PROBL ADDR INTO TCBX
05             SET RC=0 = DON'T EXECUTE EPILOG
05             * NEXT REQUEST IS FOR SAME ROUTINE
05             RETURN
04           ENDIF SPECIAL ID
03         ELSE - DIFFERENT LCB = DIFFERENT ROUTINE
04           SET RC=4 = EXECUTE EPILOG
04           * NEXT REQUEST IS FOR DIFFERENT RCUTINE
04           RETURN
03         ENDIF SAME LCB
02       ENDDO WHILE WQE PRESENT AND NO CPATCH U
02       IF NO DPATCH OF ANY KIND
03         IF ECB ADCR WAS SPECIFIED WITH PATCH
04           POST USER'S ECB
04           CLEAR ECB ADDR FRCM WQE TO
04           * PREVENT A SECOND PCST CUT OF DPPTWQDL
03         ENDIF
03         SET FLAG TASK DORMANT
03         WAIT FOR NEXT PATCH, CPATCH OR REPATCH
03         TURN OFF TASK DORMANT FLAG
02       ENDIF
01     ENDDC UNTIL ANY DPATCH
     ENDIF NC PURGE AND NO DELETE
     SET RC=4 = EXECUTE EPILOG
     * BECAUSE MODULE IS TO BE DELETED
     RETURN
```

Figure 3-176 (1 of 2). DPTPMON3

```
   BGNSEG DPTPMCN3 - USER ROUTINE LOAD
01   CLEAR RETURN CODE REG
01   GET LCB PCINTED TC BY CURRENT WQE
01   IF EP NAME BLANKS AND
01   IF SPECIAL ID SPECIFIED
02     SET DELETE FLAG TO CAUSE
02     * DPPTWQDL TO FREE THE LCB
01   ELSE
02     IF LOAD MODULE PURGE RECUESTED
03       WAIT FOR DPPTDLMP
03       CLEAR THE ECB
02     ENDIF
02     IF EP OF THIS LCB NOT RESOLVED
03       GET FIRST LCB ON TMCT-LCB CHAIN
03       STRTSRCH WHILE MORE LCB'S ARE CHAINED
03       EXITIF LCB WITH SAME NAME FCUND
04         IF LCAD MCCULE PURGE REQUC
05           WAIT FCR CPPTDLMP
05           CLEAR THE ECB
04         ENDIF
04         IF LCB MARKED FOR DELETE
05           TURN FLAG OFF TO SAVE IT
04         ENDIF
04         CHAIN TCBX-LCB TO TMCT-LCB
04         GET ENTRY POINT ADDR FRCM TMCT-LCB
04         GET ATTRIBUTE BYTE
04         CLEAR UNRESOLVED FLAG
04         INCREMENT USECOUNT IN TMCT-LCB
03       ORELSE
04         GET NEXT LCB IN TMCT-LCB CHAIN
03       ENDLOOP - EP NAME NCT FOUND ON TMCT-LCB CHAIN
04         BLDL TABLE-CET
04         CLEAR GOTTEN VIRTUAL STORAGE
04         STORE ADCR CF BLDL TBL INTO LCB
04         MOVE FFLL FIELD INTC BLCL TABLE
04         MOVE EP NAME INTO BLDL TABLE
04         ISSUE BLDL SVC
04         IF NZERO RETURN CCDE PASSED BY BLDL
05           INSERT ECB CC FOR BLDL FAILED
05           SET DELETE FLAG TC CAUSE
05           * DPPTWQDL TO FREE THE LCB
05           IF MEMBER NOT FCUND
06             LOAD MSG # FOR DPP014
05           ELSE - I/C ERROR
06             LOAD MSG # FOR CPP015
05           ENDIF
```

Figure 3-176 (2 of 2). DPTPMON3

```
05             ISSUE MESSAGE
04           ELSE - BLDL WAS SUCCESSFUL
05             COPY ATTRIBUTES FROM BLDL INTO LCB
05             IF MCDULE IS REENTRANT AND
05             IF DELETE WAS NOT SPECIFIED
06               SET FLAG LCB NEEDS LCAD BY SMCN
06               INCICATE LOAD REQEST TO DPPTSMON
06               CHAIN TCBX TO SMON'S REQ CHAIN
06               POST DPPTSMON TC LOAD THE MODULE
06               WAIT UNTIL PCSTED BY DPPTSMON
06               CLEAR PCSTBIT
05             ENDIF RENT AND NO DEL
05             IF LCB STILL UNRESOLVED
06               LOAD THE MODULE HERE
06               STORE EP ADDRESS INTO LCB
06               CLEAR UNRESOLVED-FLAG
05             ENDIF
04           ENDIF BLDL
04           BLDL TABLE-FREE
04           CLEAR IT'S ADDRESS FRCM THE LCB
03         ENDSRCH
02       ENDIF LCB UNRESCLVED
01     ENDIF EP NAME BLANK AND SPEC ID
   ENDSEG DPTPMCN3
```

Figure 3-177. DPTPMON4

```
   BGNSEG DPTPMCN4 - USER ROUTINE EXEC
01   IF EP NAME NOT BLANKS
02     IF ZERO RETURN CODE FRCM DPTFMON3
03       IF GETWA AREA TO BE TRANSFERED
04         IF
04         ENDIF
03       ENDIF
03       IF NOT SPECIAL ID
04         IF PROBL WAS MOVED INTO WQE
05           LCAD ADDR OF PROBL IN WQE
04         ELSE
05           GET ADDR OF USER'S PROBL
04         ENDIF
04         STCRE PROBL ADCR INTO TCBX
04         GET PSW PATTERN FOR USER EXECUTION
04         LOAD ACCR FOR USER PARM INTERFACE
04         LOAD ENTRY POINT ADCR
04         STORE PMON'S REGS IN HIS SA
04         GC, EXECUTE USERS RCUTINE
04
04         GET ADDR OF PMON'S SAVEAREA
04         RELCAD PMCN'S REGS FROM HIS SA
04         LCAD RETURN CODE FRCM USER EXEC
04         GET SVC STATUS
03       ENDIF SPECIAL ID
03       IF MODULE IS NCT REUSABLE OR
03       IF DELETE WAS SPECIFIED AND
03       IF MODULE WAS LOADED BY THIS TASK
04         SET LCB UNRESOLVED
04         CLEAR EP ADDRESS IN LCB
04         DELETE THE MODULE
03       ENDIF
02     ENDIF RC ZERO
01   ENCIF EP NAME BLANKS
01   STORE ECB CCMPLETICN CODE
   ENDSEG CPTPMON4
```

Figure 3-178. DPTPMON5

```
   BGNSEG DPTPMON5 - WQE-CLEANUP
01   GET ADDR OF LCB
01   IF LOAD MODULE PURGE REQUESTED AND
01   IF ECB ADDR SUPPLIED BY DPPTDLMP
02     POST DPPTDLMP'S ECB
02     CLEAR ECB ADDR FROM LCB
01   ENDIF LCBFLMP
01   GET ADDR OF AP-TYPE DUMMY GFBE
01   IF DUMMY GFBE DOESN'T POINT AT ITSELF
02     FREEWA ALL AP-TYPE GETWA AREAS
01   ENDIF
01   EXEC WQE-DELETE SVC ROUTINE
01   RESET WQ ACTIVE FLAG
   ENDSEG DPTPMON5
```

FIGURE 3-178.1. DPTPMON6

```
    BGNSEG DPTPMON6 - QP/QH INTERFACE
01  IF THIS QP IS BEING HELD
02     FORCE SEARCH FOR ANOTHER QP TO SERVICE OLD QH
01  ELSE
02     UNTIL AVAILABLE WORK FOUND ON A QH OR
02     UNTIL NO MORE QH'S
03           MOVE CLEAN UP WORK QUEUES TO QP FROM QH
03           MOVE AVAILABLE WORK TO QP WORK QUEUE CHAIN
02     ENDDO
02        QH IS SEQUENTIAL THEN
03        SET SEQUENTIAL SELECTED BIT
03     ENDIF
02     IF OLD QH IS NOT THE NEW QH AND
02     IF WORK AVAILABLE ON OLD QH
03        UNTIL AVAILABLE QP IS FOUND
04           POST QP TO PROCESS OLD QH
03        ENDDO
02     ENDIF
01   ENDIF
    ENDSEG DPTPMON6
```

Figure 3-179 (1 of 2). DPTPSVC1

```
     CLEAR RETURN CODE REGISTER
     USE DPPTSMON'S TCBX
     IF PROBL ADDR INVALID OR
     IF PRCBL LENGTH LT 4
01     SET RC FOR INVALIC PROBL
     ELSE
01     IF SUPL ADDR INVALID
02       SET RC FOR INVALID SUPL
01     ELSE
02       IF TCBX SPECIFIED
03         IF NO
04           IF NO
05             SET RC FOR INVALID PTN
04           ELSE
05             IF
06               SET RC FOR INVALID SUPL
05             ENDIF
04           ENDIF
03         ENDIF
02       ENDIF
02       IF
03         IF ECB OR TCBX ADDR INVALID
04           SET RC FOR INVALID SUPL
03         ELSE
04           CLEAR PSVC INTERNAL FLAGS
04           IF PTN=SLAVE REQUESTED AND
04           IF NOT RUNNING TWO-PARTN-OP
05             SET RC FOR INVALID PTN
04           ELSE
05             SET FIND-LOOP COUNTER TO 1
05             IF RUNNING TWO-PARTN-OP
06               IF PTN=FIND
07                 SET INTERNAL FIND FLAG
07                 SET FIND-LOOP COUNTER TO 2
06               ELSE
07                 IF PTN=MASTER OR SLAVE SPEC
08                   IF TARGET PARTN NE OWN PARTN
09                     SET OTHER PARTN FLAG
08                   ENDIF
07                 ENDIF
06               ENDIF
05             ENDIF
05             IF FREE=P WAS SPECIFIED
06               USE PROBL ADDR AND LENGTH
05             ELSE
06               USE SPEC ADDR AND LENGTH
05             ENDIF
05             IF TARGET IS OTHER PARTITION
06               SEE IF OTHER PTN IS STILL ACTIVE
06               IF IT'S ACTIVE
07                 SWITCH TO CTHER PARTITION
07                 GET ITS SCVT AND TMCT ADDR
06               ENDIF
05             ENDIF
05             IF NO ERRORS YET
06               IF FREE= WAS SPECIFIED
07                 IF TARGET IS CTHER PTN AND
07                 IF REPATCH OPTION SPEC
```

Figure 3-179 (2 of 2). DPTPSVC1

```
08                 SET RC FCR INVALID FREE
07               ELSE
08                 CFECK BEGIN ADDR OF FREE AREA
08                 IF LENGTH SPECIFIED
09                   CHECK END ADDR OF FREE AREA
08                 ENDIF
08                 IF FREE AREA IS INVALID
09                   SET RC FOR INVALID FREE
08                 ELSE
09                   IF FREE ADDR NCT ON DW BOUNDARY
09                   IF AND NOT FREEWA REQUEST
10                     SET RC FOR INVALID FREE
09                   ELSE
10                     STRTSRCH
10                     EXITIF AREA TO FREE IS
10                     EXITIF A GETWA AREA
11                       IF
11                       IF USER DID GOOD
12                         IF COULD NOT TRANSFER IT
12                         ENDIF
11                       ELSE
11                       ENDIF
10                     ENDLOOP
11                       IF USER IS ASKING US
11                       IF TO TRANSFER GETMAIN
11                       ENDIF
10                     ENCSRCH
09                   ENDIF DW BOUNCARY
08                 ENDIF FREE AREA WITHIN PARTITION
07               ENDIF CTHER PTN AND REPATCH
06             ENDIF FREE SPECIFIED
05           ENDIF
04         ENDIF SLAVE REQUESTED BUT NOT INIT
03       ENDIF ECB OR TCBX INVALID
02     ENDIF
01   ENDIF SUPL INVALID
   ENDIF PROBL INVALID
```

Figure 3-180. DPTPSVC2

```
   BGNSEG ADRCHK - ADDRESS CHECK (CWN + OTHER PTN)
01   IF INPUT NCT AN ADDR WITHIN OWN PTN
02     IF TWO-PARTN-OP INITIALIZED
03       IF INPUT NCT AN ADDR WITHIN OTHER PTN
04         SET NCNZERO RETURN CODE
03       ENDIF
02     ELSE
03       SET NCNZERO RETURN CODE
02     ENDIF SCVTF2PT
01   ENDIF
   ENDSEG ADRCHK


   BGNSEG ADRCHK1 - ACCRESS CHECK (OWN PTN)
01   IF INFUT NCT AN ACDR WITHIN OWN PTN
02     SET NCNZERO RETURN CODE
01   ENDIF
   ENDSEG ADRCHK1
```

Figure 3-181 (1 of 3). DPTPSVC3

```
     BGNSEG DPTPSVC3 - BUILD WQE AND LCB
01     IF TASK IS BEING REMOVED (DPATCHEC)
02       SET RETURN CODE DPATCH IN PROGRESS
01     ELSE
02       IF NOT QPOS=DPATCH
03         IF LIMIT QUEUE LENGTH NOT ZERO
04           IF CURRENT QUEUE LENGTH GE LIMIT
05             IF QPOS=FIRST
06               SET RC WQE LOST WITH QPOS=FIRST
05             ELSE
06               SET RC FOR WORK QUEUE FULL
05             ENDIF
04           ENCIF CQL GE LQL
03         ELSE - LIMIT CUEUE LENGTH IS ZERO
04           IF CURRENT WQ ACTIVE
05             SET RC FOR WORK QUEUE FULL
04           ELSE
05             SET CURRENT WQ ACTIVE FLAG
04           ENDIF
03         ENDIF LQL ZERC
02       ELSE - CPOS DPATCH
03         IF DPATCH CUEUE IS EMPTY (HAS 1 SLOT ONLY)
04           GET DPATCH WQ ORIGIN
03         ELSE
04           SET RC FOR DPATCH QUEUE FULL
03         ENDIF
02       ENDIF QPOS DPATCH
02       IF FREE=P WAS SPECIFIED
03         GET ADDR AND LENGTH FROM PROBL
02       ELSE
03         GET ACDR ANC LENGTH FRCM SUPL
02       ENDIF
02       IF RETURN CODE LOW
03         CB-GET FOR WQ-ELEMENT
03         IF CB-GET PASSED NCNZERO RETURN CODE
04           SET RC FCR NO CB-GET STORAGE
03         ELSE
04           FILL IN WQE FIELDS
04           STORE FREE LENGTH AND ACCR
04           MOVE FLAGS FROM SUPL
04           MOVE USERS ECB ADDR FRCM SUPL
04           MOVE USER'S ID FRCM PROBL
04           IF PROBL LIST LE 8 BYTES LONG
05             MCVE PRCBL LIST INTO WQE
05             SET FLAG PARMLIST IS IN WQE
04           ELSE
05             STORE PROBL ADCR INTO WQE
04           ENDIF
```

Figure 3-181 (2 of 3). DPTPSVC3

```
04          STRTSRCH WHILE MORE LCBS CHAINED
04          EXITIF EP NAME FOUND
04          ORELSE
04          ENDLOOP - NOT FOUND
05            CB-GET FOR LCB
05            IF CB-GET PASSED NONZERO RETURN CODE
06              CB-FREE THE WQ-ELEMENT
06              SET RC FOR NO CB-GET STORAGE
05            ELSE
06              CHAIN AT END OF TCBX-LCB CHAIN
06              SET LCB UNRESOLVED
06              MOVE EP NAME INTO LCB
05            ENDIF LCB GET
04          ENDSRCH LCB CHAIN
04          IF RETURN CCDE LOW
05            STORE CUR OWN TCB ACCR INTO WQE
05            STORE TCBX ADDR INTO WQE
05            STORE LCB ADDR INTO WQE
05            INCREMENT TCBX-LCB REQUEST CCUNT
04          ENDIF
04          IF RETURN CCDE LOW
05            IF QPOS=DPATCH
06              STORE WQE ACDR INTO TCBXDWQ
05            ELSE - NCT QPOS=DPATCH
06              WHILE MORE WQE'S ARE CHAINED
07                KEEP ADDR CF PREVIOUS WQ
07                KEEP ADDR OF THIS WQ
06              BGNWHILE
07                GET ADDR OF NEXT WCE IN CHAIN
06              ENDDO
06              IF QUEUE LENGTH NOT ZERO
07                IF CURRENT QUEUE LENGTH LT LIMIT
08                  INCREMENT CURRENT QUEUE LENGTH
08                  IF HIWATER LT CURRENT Q LENGTH
09                    STORE CURRENT AS HIWATER
08                  ENDIF
07                ELSE - CURRENT QUEUE LENGTH NOT LESS
08                  DECHAIN THE LOST WQE
08                  PUT ECB CCMPL CODE IN WQE
08                  GET ORIGIN CF CLEANUP-WQ
08                  WHILE MORE WQE'S ARE CHAINED
09                    KEEP ADDR OF THIS WQE
08                  BGNWHILE
09                    GET ADDR OF NEXT WQE IN CHAIN
08                  ENDDO
08                  CHAIN LOST WQE TO CLEANUP Q
07                ENDIF CQL LT LQL
06              ENDIF LQL NZERO
```

Figure 3-181 (3 of 3). DPTPSVC3

```
06            IF QPOS=FIRST
07              CHAIN THIS WQE AT TOP OF QUEUE
06            ELSE - NOT QPOS FIRST
07              CHAIN THIS WQE AT BOTTOM
06            ENDIF
05          ENDIF QPOS DPATCH
05          IF CHAP REQUESTED BY DPTPSVC4
06            LOAD ECB ADDR TO POST DPPTSMON
06            LOAD ADDR OF SMON'S TCB = JOBSTEP
05          ELSE - NO CHAP REQ - TASK EXISTED ALREADY
06            LOAD ECB ADDR TO POST DPPTPMON
06            LOAD ADDR OF ASSOC PMON'S TCB
05          ENDIF
05          IF ECB NOT POSTED
06            POST ECB (DPPTSMON OR DPPTPMON)
05          ENDIF
04        ENDIF RC LOW
03      ENDIF WQ GET
02    ENDIF RC LOW
01  ENDIF DPATCH IN PROGRESS
  ENDSEG DPTPSVC3
```

Figure 3-182 (1 of 2). DPTPSVC4

```
  BGNSEG DPTPSVC4 - BUILD TCBX
01  IF PRTY REFERENCE TCBXNAME SPECIFIED
02    GET INDEPENDENT TASK CHAIN ORIGIN
02    STRTSRCH WHILE MORE TCBX'S CHAINED
02    EXITIF PRTY REFERENCE TCBXNAME FOUND
02    ORELSE
03      GET ADDR OF NEXT TCBX IN CHAIN
02    ENDLOOP - TCBXNAME NOT FOUND ON CHAIN
03      SET RC FOR PRTY TCBXNAME NOT FOUND
02    ENDSRCH
01  ELSE - NO PRTY TCBXNAME SPEC
02    LOAD TCBUSER FIELD OF CURRENT TASK
01  ENDIF
01  IF RETURN CODE LOW
02    IF PATCHOR DOESN'T HAVE A TCBX
03      GET BASE PRTY VALUE FROM TCB
02    ELSE
03      GET BASE PRTY VALUE FROM TCBX
02    ENDIF
02    GET FIRST TCBX FROM TMCTFREE CHAIN
02    IF THERE IS ONE
03      DECHAIN IT FROM FREECHAIN
03      RESET IT'S HIWATER QUEUE LENGTH
03      DECREMENT # OF FREE TCBX'S
```

Figure 3-182 (2 of 2). DPTPSVC4

```
02      ELSE - NO FREE TCBX AVAILABLE
03        CB-GET FOR TCBX
03        IF NCNZERO RETURN CODE PASSED FROM CBGET
04          SET RC FOR NO CB-GET STORAGE
03        ELSE
04          SET FLAG TCB NEEDED
04          INCREMENT HIWATER # OF TCBX'S
04          MOVE XCVT ADDR INTO TCBX
04          INITIALIZE GETWA/FREEWA ORIGINS
03        ENDIF TCBX GET
02      ENDIF TCBX CN FREECHAIN
02      IF RETURN CODE LOW
03        MCVE IN TCBXNAME
03        MCVE IN QUEUE LENGTH
03        IF SUPL IS AN UNMODIFIED REPL
04          USE ABSOLUTE PRTY FROM REPL
03        ELSE - SUPL OR MODIFIED REPL
04          CALC ABSOLUTE PRTY VALUE
03        ENDIF
03        IF REQUESTED PRTY GT LIMIT PRTY
04          USE LIMIT PRTY INSTEAD
03        ELSE
04          USE REQUESTED PRTY
03        ENDIF
03        GET TMCTSMON REQUEST CHAIN ORIGIN
03        STRTSRCH WHILE MORE TCBX'S CHAINED
03        EXITIF THAT TCBX IS ALREADY QUEUED
03        ORELSE
04          GET ADDR OF NEXT TCBX IN CHAIN
03        ENDLOCP - NCT FCUND ON CHAIN
04          CHAIN TCBX TO SMON'S REQ CHAIN
03        ENDSRCH
03        DO DPTPSVC3 - BUILD WQE ANC LCB
03        IF RETURN CODE LOW
04          INCREMENT # OF ACTIVE TCBX'S
03        ELSE - DPTPSVC3 WAS NCT SUCCESSFUL
04          CLEAR TCBXNAME
04          CLEAR QUEUE LENGTH
04          CHAIN TCBX TO FREE CHAIN
04          RESET CHAP FLAG FOR ZERO PRTY
04          INCREMENT # OF FREE TCBX'S
03        ENDIF
02      ENDIF RC LOW
01    ENDIF RC LCW
  ENDSEG DPTPSVC4
```

Figure 3-183. DPTPSVC5

```
     BGNSEG   DPTPSVC5 - DETERMINE IF OTHER PTN IS STILL ACTIVE
01      LOAD   OTHER PTN'S  TCB ADDRESS FROM SCVT
01      IF     TCB ADDRESS IS ZERO OR
01      IF     OTHER TCB IS ABENDING
02        SET ERROR RETURN CODE
01      ELSE
02        IF   IT'S NOT JOB STEP
03          SET ERROR RETURN CODE
03          TURN OFF TWO PTN FLAGS
02        ENDIF
01      ENDIF
     ENDSEG   DPTPSVC5
```

Figure 3-184. DPTSMON1

```
    IF LOAD REQUEST FLAG SET
01    GET TCBX-LCB CHAIN ORIGIN
01    STRTSRCH WHILE MORE LCB'S ON TCBX CHAIN
01    EXITIF LCB WITH LOAD REQUEST FOUND
02      GET TMCT-LCB CHAIN ORIGIN
02      STRTSRCH WHILE MORE LCB'S ON CHAIN
02      EXITIF LCB WITH SAME NAME
03        STORE TMCT-LCB ADDR INTO TCBX-LCB
03        COPY EP ADDR FROM TMCT-LCB TO TCBX-LCB
03        COPY ATTRIBUTE BYTE
03        CLEAR FLAGS
03        INCREMENT USECOUNT IN TMCT-LCB
02      ORELSE
03        GET NEXT LCB ON TMCT-LCB CHAIN
02      ENDLOOP - NOT FOUND ON TMCT-LCB CHAIN
03        CB-GET FOR LCB
03        IF CB-GET RETURNED ZERO RETURN CODE
04          STORE TMCT-LCB ADDR INTO TCBX-LCB
04          USE BLDL-LIST GOTTEN BY DPPTPMON
04          LOAD THE MODULE
04          STORE EP-ADDRESS INTO TCBX-LCB
04          CLEAR FLAGS FROM TCBX-LCB
04          CHAIN NEW LCB TO TMCT-LCB CHAIN
04          SET FLAG LCB ON TMCT CHAIN
04          COPY EP ADDR + NAME INTO TMCT-LCB
04          COPY ATTRIBUTES INTO TMCT-LCB
04          INCREMENT USECOUNT IN TMCT-LCB
03        ENDIF CBGET ZERO RETCODE
02      ENDSRCH TMCT-LCB CHAIN
01    ORELSE
02      GET NEXT LCB ON TCBX-LCB CHAIN
01    ENDLOOP - NO LOAD REQUEST FOUND ON TCBX-LCB CHAIN
02      ISSUE MESSAGE DPP017 - NO LOAD REQ FOUND
01    ENDSRCH TCBX-LCB CHAIN
01    CLEAR LOAD REQUEST FLAG FROM THE TCBX
01    POST DPPTPMON TO RESUME PROCESSING
    ENDIF LOAD REQ
```

Figure 3-185. DPTWSVC1

```
   DPTWSVC1 - INCLUDED SEGMENT - GETWA BLOCK GET ROUTINE
   GET CURRENT # FREE BLOCKS
   IF THERE ARE BLOCKS AVAILABLE
01   FINC A GFCB WITH BLOCKS AVAILABLE
01   TURN CFF GFCB'S ALL BLKS FREE FLG
01   DECREMENT THE GFCB'S FREE COUNT
01   CET ADDR CF FREE GFBE
01   ALLOCATE GFBE AND TAKE IT OFF THE FREE CHAIN
01   UPCATE FREELIST PCINTER
01   ALLOCATE THE GFBE
01   CHAIN THE ALLOCATED GFBE TO THE PROPER CHAIN - AT,AP,OR PC
   ELSE
01   PASS A NEGATIVE RETURN CCDE
01   GET TYPE AND LENGTH CODES IN HI-ORD BYTE REG 1
   ENDIF
```

Figure 3-186. DPTWSVC3

```
   DPTWSVC3 - INCLUDED SEGMENT - BRANCH ENTRY ROUTINE
   WHILE NEXT PCINTER IS ZERO - DO
01   GET ASSOCIATEC GFMB'S ID
01   ID X GFMBLNTH = GFMB OFFSET
01   CALC ADDR OF ASSOCIATED GFMB
01   GET ASSOCIATEC GFCB ADDR
01   LOCATE THE GFCB TO WHICH THIS GFBE BELCNGS
01   INCREMENT THE GFCB FREE CCUNT
01   IF NOT INITIAL ALLOCATION
02     IF ALL BLCCKS IN THIS GFCB ARE FREE
03       TURN CN THE ALL FREE FLAG
03       IF GFCB IS NCT LAST ON THE CHAIN
04         MOVE THE GFCB TO THE END OF THE GFMB CHAIN
03       ENDIF
02     ENDIF
01   ENDIF
01   TURN OFF ALLOCATEC BIT
01   GET PTR TC NEXT CFBE CN CHAIN
01   GET PTR TO FIRST FREE GFBE
01   CLEAR HI-ORDER BYTE (ALLOCBIT)
01   RETURN CURRENT GFBE TO FREE CHN
01   CCMPLETE FREE CHAIN
01   GET FREE CCUNT AND
01   INCREMENT IT ANC
01   RESTORE THE INCREMENTED CCUNT
    ENDDO
    RETURN
```

Figure 3-187. DPXDBIN1

```
   DPXDBIN1-INCLUDED SEGMENT            OBTAIN CORE FOR @INIT DATA
01   OPEN DATA BASE DCB (PARTITIONED)
01   UNTIL ALL MEMBERS HAVE BEEN PROCESSED
02     COUNT THE NUMBER OF CA RESIDENT ARRAYS
02     COUNT THE NUMBER OF LOG ARRAYS
01   ENDDO
01   USE ARRAY ID OF LAST MEMBER AS THE TOTAL NUMBER OF ARRAYS
01   GETMAIN FOR CONTROL BLOCK CORE BASED ON NUMBER OF ARRAYS,LOG ARRAYS,
   *                CA RESIDENT ARRAYS
   ENDSEGMENT DPXDBIN1
```

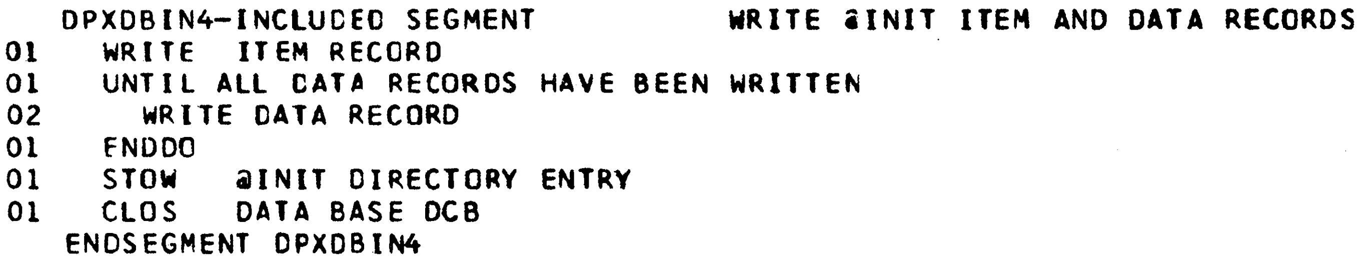
Figure 3-188. DPXDBIN2

```
   DPXDBIN2-INCLUDED SEGMENT      .     BUILD CONTROL BLOCKS
01   UNTIL ALL MEMBERS HAVE BEEN PROCESSED
02     BUILD DUMMY ALT'S FOR ALL NUMBERED ARRAYS THAT WERE SKIPED
02     BUILD PRIMARY ALT FROM DIRECTORY ENTRY
02     BUILD SECONDARY ALT FROM DIRECTORY ENTRY
02     IF IT'S NOT A DUMMY ARRAY
03       IF IT'S CA RESIDENT ARRAY
04         READ IN FIRST DATA RECORD
04         IF DCB HAS NOT BEEN BUILT FOR THIS DDNAME
05           BUILD DCB
04         ENDIF
03       ELSE
04         IF IT'S A LOGGED ARRAY
05           READ IN FIRST DATA RECORD
05           BUILD LOG CONTROL BLOCK
04         ENDIF
03       ENDIF
02     ENDIF
01   ENDDO
01   BUILD DCB FOR DATA BASE INITIALIZATION DATA SET
01   BUILD DCB FOR COMPOSITE ITEMS DATA SET
   ENDSEGMENT DPXDBIN2
```

Figure 3-189. DPXDBIN3

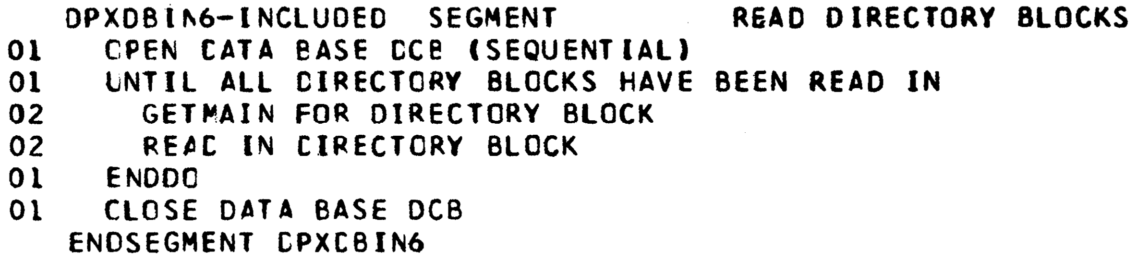
```
    DPXDBTN3-SORT VS RESIDENT ARRAY DATA ADCRESSES
01      UNTIL EACH USE CCDE HAS EEEN PROCESSED
02        GETMAIN  A SORT TABLE FOR THIS USE CODE
01      ENDDO
01      UNTIL ALL VS RESIDENT ARRAYS WHICH REQUEST PAGE BOUNDARIES ARE
   *                                         PROCESSED
01        FIND SORT TABLE FOR THIS ARRAY'S USE CODE
01        FIND NEXT AVAILABLE SLOT
01        SAVE ARRAY ID IN SLOT
01        SAVE THE RELATIVE PAGE NUMBER FOR THIS USE CODE IN SLOT
01        SAVE ARRAY SIZE IN THE PALT DATA FIELD
01        SAVE THE NUMBER OF BYTES LEFT IN THE PAGE AT THE END OF THIS ARRAY
        ENDDC
        UNTIL ALL VS RESIDENT ARRAYS WHICH DO NOT REQUEST PAGE BOUNDARIES
   *                                         ARE PROCESSED
01        FIND NEXT AVAILABLE SLOT
01        SAVE ARRAY IC IN SLOT
01        SAVE THE ARRAY SIZE IN THE PALT CATA FIELD
01        WHILE ANY PREVIOUSLY USED SLOTS HAVE NOT BEEN EXAMINED OR
01        UNTIL TFIS ARRAY'S DATA WILL FIT INTO A PREVIOUSLY USED PAGE
02          GET NEXT SLOT
01        ENCDO
01        IF  NO FIT WAS FOUND AND
01        IF  MIKIMUM BOUNDARY REQUESTED
02          SAVE THE RELATIVE PACE NUMBER FOR THIS USE CODE IN THE SLOT
01        ENDIF
01        RESET NC.OF BYTES LEFTOVER IN THIS PAGE
01        INCREMENT THE COUNT OF ARRAYS FOR THIS PAGE
01        SAVE COUNT CF ARRAYS IN SLOT
        ENDDC
        UNTIL EACH USE CCDE SORT TABLE HAS BEEK PROCESSED
01        SORT THE SORT TABLE BASED ON RELATIVE PAGE NUMBER AND COUNT NUMBER
   *                USINC A MERCE EXCHANGE TECFNIQUE
01      ENDDO
01      UNTIL EACH USE CODE SORT TABLE HAS BEEN PROCESSED
02        UNTIL EACH SLOT IN THE SORT TABLE HAS BEEN PROCESSED
03          IF PAGE BOUNCARY NEECED
04            RCUND RELATIVE DATA ADDRESS TO NEXT PAGE BOUNDARY
03          ENDIF
03          SAVE RELATIVE DATA ADDRESS IN PACT CATA FIELD
03          INCREMENT RELATIVE CATA ACCRESS BY SIZE OF THIS ARRAY
02        ENDDC
01      ENDDO
     ENDSEGMENT
```

Figure 3-190. DPXDBIN4



```
   DPXDBIN4-INCLUDED SEGMENT          WRITE @INIT ITEM AND DATA RECORDS
01   WRITE  ITEM RECORD
01   UNTIL ALL CATA RECORDS HAVE BEEN WRITTEN
02     WRITE DATA RECORD
01   ENDDO
01   STOW   @INIT DIRECTORY ENTRY
01   CLOS   DATA BASE DCB
   ENDSEGMENT DPXDBIN4
```

Figure 3-191. DPXDBIN6



```
   DPXDBIN6-INCLUDED  SEGMENT          READ DIRECTORY BLOCKS
01   CPEN CATA BASE CCB (SEQUENTIAL)
01   UNTIL ALL CIRECTORY BLOCKS HAVE BEEN READ IN
02     GETMAIN FOR DIRECTORY BLOCK
02     REAC IN CIRECTORY BLOCK
01   ENDDO
01   CLOSE DATA BASE DCB
   ENDSEGMENT CPXCBIN6
```

Appendix A. DIRECTORY

This appendix contains cross-references of the Special Real Time Operating System to the following:

- CSECT Names/Source Members
- Module Names/HIPO-PDL Charts
- Macro Names/Module Names
- Operator Command/Module Name
- Module Name/Functional Area

CSECT NAMES/SOURCE MEMBERS

This subsection contains a cross-reference of Special Real Time Operating System CSECT names and source members. The CSECTS (Load Modules) are named according to the following standards. The first three characters are either DOM or DPP as defined by IBM product code standards. Those CSECTS that begin with DOM are associated with the failover/restart function. Those CSECTS that begin with DPP utilize the fourth character to denote the section of the system where they belong, as follows:

DPP - First three characters
Fourth character
C - Time Management
D - Online Data Base
F - FORTRAN Interface Routines
I - Initialization Routines
M - Message Handler
P - PL/I Interface Routines
S - Duplicate Data Set Support
T - Task Management
U - Offline Programs
X - Miscellaneous
Z - Test and Diagnostic Program

The remaining four characters are intended to have meaning to identify the function of the CSECT.

The copied segments follow a similar naming convention in most cases. The difference is that one "P" is omitted from the CSECT name and a numeric digit is added. For example, DPCTIME1 and DPCTIME2 are copied segments that are a part of DPPCTIME.

The exception to the above convention is the CSECT IEAXYZ5. This CSECT is included as a part of the OS/VS1 fixed nucleus and, as such, must conform to the requirements of OS/VS1.

CSECT NAME	MAIN SEGMENT	COPIED SEGMENT	
DOMICEXT	DOMICEXT		SUBSTITUTE EXTERNAL FLIH
IEAXYZ5	DOMICEXT		ALTERNATE NAME FOR ABOVE MODULE
DOMIRBT	DOMIRBT		FAILOVER/RESTART BOOTSTRAP
		DOMIRSIO	STAND ALONE I/O ROUTINE
DOMIRCMN	DOMIRCMN		CONTINUOUS MONITOR
DOMIRCPY	DOMIRCPY		COPY A FAILOVER/RESTART DATA SET
DOMIRFLV	DOMIRFLV		LOAD 1 F/R SVC
DOMIRFL2	DOMIRFL2		LOAD 2 F/R SVC
DOMIRINT	DOMIRINT		F/R-EXTERNAL INTERRUPT INIT.
DOMIRNIP	DOMIRNIP		RE-NIP
		DOMIRSIO	STAND ALONE I/O ROUTINE
DOMIRPRB	DOMIRPRB		PROBE
DOMIRWT	DOMIRWT		FAILOVER/RESTART WRITE
		DOMIRSIO	STAND ALONE I/O ROUTINE
			THE FOLLOWING 3 MODULES ARE NAMED AT SYSGEN TIME ACCORDING TO NUMBERS SUPPLIED
***	DOMISVC1		TYPE 1 SVC PREFIX HANDLER
***	DOMISVC2		TYPE 2 SVC PREFIX HANDLER
***	DOMISVC4		TYPE 4 SVC PREFIX HANDLER
DOMXLIST	DOMXLIST		PREPARE IEHLIST INPUT
DOMXSTG1	DOMXSTG1		STAGE 1 OF SYSGEN UTILITY
		MXSTG101	
		MXSTG102	
		MXSTG103	
		MXSTG104	
		MXSTG105	
DPPCALCF	DPPCALCF		CALCULATE CORRECTION FACTOR
		DPCALCF1	OBTAIN BASE TIME
DPPCPTIM	DPPCPTIM		PTIME MONITER ROUTINE
DPPCTIME	DPPCTIME		TIME UPDATE ROUTINE
		DPCTIME1	POST MIDNIGHT TIME CORRECTION
		DPCTIME2	TIME ERROR ROUTINE
DPPCTSVC	DPPCTSVC		PTIME TYPE 2 SVC
		DPCTSVC1	RET OPTION
		DPCTSVC2	ADD OPTION
		DPCTSVC3	MOD OPTION
		DPCTSVC4	DELETE OPTION
DPPCUPCF	DPPCUPCF		UPDATE CORRECTION FACTOR
		DPCUPCF1	UPDATE CORRECTION FACTOR
		DPCUPCF2	UPDATE TIME
		DPCUPCF3	UPDATE DATE
		DPCUPCF4	UPDATE PTQE S
DPPDARAY	DPPDARAY		GET PUTARAY PROCESSOR
DPPDBLOK	DPPDBLOK		GET PUT BLOCK SUBROUTINE
DPPDBSIF	DPPDBSIF		TWO PARTITION INTERFACE FOR DATA BASE(M)
DPPDFREQ	DPPDFREQ		CYCLIC LOGGING ROUTINE
DPPDGETL	DPPDGETL		GETLOG ROUTINE

CSECT NAME	MAIN SEGMENT	COPIED SEGMENT	
DPPDITEM	DPPDITEM		GET PUTITEM PROCESSOR
DPPDPUTL	DPPDPUTL		PUTLOG ROUTINE
DPPDRIFE	DPPDRIFE		DUMMY INIT.TIME SETTER
DPPDRIFT	DPPDRIFT		TIME DRIFT CORRECTION
DPPDSUB2	DPPDSUB2		SLAVE PARTITION INTERFACE ROUTINE
DPPDUMPL	DPPDUMPL		DUMPLOG ROUTINE
DPPDUPDL	DPPDUPDL		LOGGING REFRESH ROUTINE
DPPDWRST	DPPDWRST		DATA BASE OPEN/CLOSE ON WRITE/RESTART
DPPFAUNC	DPPFAUNC		FORTRAN SUBROUTINE FOR COPY ADDR BIT SET
DPPFIXFR	DPPFIXFR		PAGE FIX/FREE HANDLER
DPPIDBAS	DPPIDBAS		DATA BASE INITIALIZATION
		DPIDBAS1	READ DB TABLES INTO VS STORAGE
		DPIDBAS2	INITIALIZE DB TABLES
		DPIDBAS3	READ VS RESIDENT ARRAYS INTO STORAGE
		DPIDBAS6	INITIALIZATION FOR SLAVE PARTITION
DPPIIRE	DPPIIRE		SCHEDULE DB OPEN/CLOSE ON WRITE/RESTART
DPPILOGN	DPPILOGN		LOGGING INITIALIZATION
DPPINIT	DPPINIT		INITIALIZATION TASK MANAGEMENT
		DPINIT1	CREATE TMCT INITIALIZE GETWA
		DPINIT2	INITIALIZE CBGET
		DPINIT3	CREATE ADVANCE TCB S
		DPINIT5	SYNCHRONIZE FOR TWO PARTITION
DPPINIT0	DPPINIT0		INITIALIZATION CARD READ ROUTINE
		DPINIT01	PARAM KEYWORD PROCESSOR
		DPINIT02	PATCH CARD PROCESSOR
		DPINIT03	CONTINUATION CARD PROCESSOR
		DPINIT04	CONTROL CARD PROCESSOR
		DPINIT05	BUILD WAIT LIST ROUTINE
		DPINIT06	ROUTINE TO HANDLE BLANKS IN PARAM
		DPINIT08	CONTROL BLOCK BUILD AND CHAIN
		DPINIT09	ERROR MESSAGE WRITER
		DPINIT0A	GP/GH STAEX CARD PROCESSOR
DPPINIT1	DPPINIT1		INITIALIZATION SUBSYSTEM PATCHOR
		DPINIT11	ERROR MESSAGE ROUTINE
		DPINIT12	PROBE INTERFACE
		DPINIT13	CONTINUOUS MONITOR INTERFACE
		DPINIT14	PROBE/TIME IRB TO SUPPRESS TIME INTERRUPTS
DPPIPFIX	DPPIPFIX		PAGE FIX ROUTINE
DPPIPFRE	DPPIPFRE		PAGE FREE UNFIX ROUTINE
DPPISTAE	DPPISTAE		JOB STEP TASK STAE ROUTINE
DPPITIM1	DPPITIM1		TIME INITIALIZATION
		DPITIM11	INITIALIZE TIME IN TIME ARRAY
DPPMINIT	DPPMINIT		MSG HANDLER INITIALIZATION
DPPMMSG	DPPMMSG		SYSTEM MESSAGE FORMATTER
DPPMMSGV	DPPMMSGV		SYSTEM MESSAGE ROUTING CODE CHANGE
DPPMMSG1	DPPMMSG1		SYSTEM MESSAGE OUTPUT ROUTINE
DPPPARM	DPPPARM		PL/I INTERFACE ROUTINE
DPPPIF	DPPPIF		PL/I INTERFACE ROUTINE
DPPSAMP1	DPPSAMP1		SAMPLE PROGRAM-PATCH ROUTINE
DPPSASOC	DPPSASOC		DDS ASYNCHRONIS OPEN OR CLOSE
		DDSDSECT	DSECT OF DDS CONTROL AREA

CSECT NAME	MAIN SEGMENT	COPIED SEGMENT	
DPPSBFST	DPPSBFST		BLDL FIND TYPE D STOW FOR A DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSBF1	DPPSBF1		BLDL FIND TYPE D FOR A DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSCHCK	DPPSCHCK		DDS CHECK MODULE
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSCHK2	DPPSCHK2		CHECK A DDSDECB INTERNAL
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSCHK3	DPPSCHK3		SYNAD ROUTINE FOR DDS CHECK
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSCHK4	DPPSCHK4		CHECK A BACKUP DECB
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSCHPR	DPPSCHPR		SET A PRIMARY DECB AND A BACKUP DECB
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSCLUP	DPPSCLUP		DDS CLEAN UP ROUTINE
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSCL1	DPPSCL1		CLOSE A DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSCMPR	DPPSCMPR		COMPARE FOR DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSCP2B	DPPSCP2B		COPY A DDS PRIMARY TO BACKUP
DPPSCRBK	DPPSCRBK		CREATE A DDS BACKUP
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSCT2T	DPPSCT2T		COPY TRACK TO TRACK
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSDDSX	DPPSDDSX		SEARCH FOR AN IOA FROM A GIVEN DDCB
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSINIT	DPPSINIT		INITIALIZE THE DDS SYSTEM
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSIN12	DPPSIN12		PROCESS THE CTLIN STREAM
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSIN13	DPPSIN13		VALIDATE EACH DDS DECLARATION WRT JCL
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSIN14	DPPSIN14		ALLOCATE AND INITIALIZE PERMANENT DDSCTLA
		DDSNTPT	COPY CODE FOR NOTE POINT BRANCH
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSIN15	DPPSIN15		DEFORMAT THE CTLIN CARD
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSIN16	DPPSIN16		DEFINE LOCKS FOR EACH CTLA
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSLOCK	DPPSLOCK		LOCK A DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSMSGI	DPPSMSGI		DDS INPUT MESSAGE PROCESSOR
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSMSGO	DPPSMSGO		DDS MESSAGE OUTPUT PROCESSOR
DPPSNOTE	DPPSNOTE		PERFORM NOTE ON A DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA

CSECT NAME	MAIN SEGMENT	COPIED SEGMENT	
DPPSNTPT	DPPSNTPT		BRANCH CODE FOR NOTE POINT
		DDSNTPT	COPY CODE FOR NOTE POINT BRANCH
DPPSOPCL	DPPSOPCL		OPEN CLOSE HALF OF A DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSOP1	DPPSOP1		OPEN A DDSDCB
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSOP2	DPPSOP2		OPEN A DECLARED DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSPNTF	DPPSPNTF		PERFORM POINT FIND TYPE C ON A DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSRCIO	DPPSRCIO		RECREATE I O FOR A DDS HALF
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSRDWT	DPPSRDWT		READ WRITE MODULE FOR DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSRDW2	DPPSRDW2		ACTUAL READ WRITE FOR A DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSRLSE	DPPSRLSE		RELEASE A DDSDECB
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSRSRV	DPPSRSRV		RESERVE A DDSDECB
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSRSTR	DPPSRSTR		DDS FAILOVER/RESTART
DPPSRTCP	DPPSRTCP		DDS REAL TIME COPY
DPPSSHAR	DPPSSHAR		SHARE A DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSSRCH	DPPSSRCH		SEARCH A FIXED LENGTH TABLE FOR AN ENTRY WHO
DPPSST1	DPPSST1		STOW FOR A DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSSWCH	DPPSSWCH		SWITCH PRIMARY TO BACKUP FOR A DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSTBOS	DPPSTBOS		TAKE A BACKUP OUT OF SERVICE
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSUNLK	DPPSUNLK		UNLOCK A DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSUNSH	DPPSUNSH		UNSHARE A DDS
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPSWRST	DPPSWRST		DDS WRITE STATUS
DPPSXTCB	DPPSXTCB		LOCATE ALLOCATE A DDSXTCBC FOR AN INPUT TASK
		DDSDSECT	DSECT OF DDS CONTROL AREA
DPPTCBGT	DPPTCBGT		CBGET TYPE 1 SVC ROUTINE
DPPTCSVC	DPPTCSVC		CHAIN TYPE 1 SVC ROUTINE VALIDITY
		DPTCSVC1	CHAIN FUNCTION
DPPTDLMP	DPPTDLMP	DPPTDLMP	LOAD MODULE PURGE REENTRANT
		DPTDLMP1	TCBX LCB USE SCAN
		DPTDLMP2	TCBX LCB PURGE SCAN
		DPTDLMP3	TCBX POST SCAN
		DPTDLMP4	STIMER EXIT RTN
		DPTDLMP5	ASYNC DELETE RTN
DPPTDSVC	DPPTDSVC	DPPTDSVC	DPATCH SVC RTN TYP 1 SVC RTN
		DPTDEBUG	DEBUG AID
		DPTDSVC1	VALIDITY CHECK
		DPTECBCC	ECB CC DEFINITIONS
		DPTPSVC2	ADDRESS CHECK

CSECT NAME	MAIN SEGMENT	COPIED SEGMENT		
DPPTETXR	DPPTETXR	DPPTETXR	END OF TASK EXIT	REENTRANT
	DPPTETXM	DPPTETXM	WTO ETXRMSG	
		DPTECBCC	ECB CC DEFINITIONS	
DPPTGWFW	DPPTGWFW		GETWA/FREEWA BRNCH ENTRY SUBROUTINE	
DPPTIMPS	DPPTIMPS		STAE-IMP COMMAND PROCESSING	
DPPTPMON	DPPTPMON	DPPTPMON	PATCH MONITOR	REENTRANT
		DPTECBCC	ECB CC DEFINITIONS	
		DPTPMON1	TCBX CLEANUP	
		DPTPMON2	HI LVL LANG ENTRY	
		DPTPMON3	USER ROUTINE LOAD	
		DPTPMON4	USER ROUTINE EXEC	
		DPTPMON5	WQE CLEANUP	
		DPTPMON6	QP/QH INTERFACE	
DPPTPSVC	DPPTPSVC	DPPTPSVC	PATCH SVC RTN	TYP 1 SVC RTN
		DPTDEBUG	DEBUG AID	
		DPTECBCC	ECB CC DEFINITIONS	
		DPTPSVC1	VALIDITY CHECK	
		DPTPSVC2	ADDRESS CHECK	
		DPTPSVC3	BUILD WQE LCB	
		DPTPSVC4	BUILD TCBX	
		DPTPSVC5	TEST OTHER PARTITION TO SEE IF ITS ACTIVE	
DPPTPWQE	DPPTPWQE		PURGE WORK QUEUE	
DPPTQIMP	DPPTQIMP		IMP QS COMMAND PROCESSOR	
DPPTRGWA	DPPTRGWA		TRANSFER GETWA AREA	
DPPTRSVC	DPPTRSVC	DPPTRSVC	REPATCH SVC RTN	TYP 2 SVC RTN
		DPTDEBUG	DEBUG AID	
		DPTECBCC	ECB CC DEFINITIONS	
		DPTPSVC2	ADDRESS CHECK	
DPPTSMON	DPPTSMON	DPPTSMON	SYSTEM MONITOR	REENTRANT
		DPTECBCC	ECB CC DEFINITIONS	
		DPTSMON1	LOAD RENT MODULES	
DPPTSTAE	DPPTSTAE		STAE-EXIT ROUTINE FOR STAE/IMP COMMAND	
DPPTWQDL	DPPTWQDL	DPPTWQDL	WQE DELETE RTN	TYP 2 SVC RTN
		DPTDEBUG	DEBUG AID	
		DPTECBCC	ECB CC DEFINITIONS	
DPPTWSVC	DPPTWSVC		GETWA FREEWA TYPE 1 SVC ROUTINE	
		DPTWSVC1	GETWA ROUTINE	
		DPTWSVC3	BRANCH ENTRY ROUTINE	
DPPUMSG	DPPUMSG		DEFINE MESSAGE	
		DPPUMSG1	DELETES MESSAGES	
		DPPUMSG2	ADDS,TESTS,REPLACES MESSAGES	
DPPXDBAS	DPPXDBAS		DATA BASE FINAL PHASE PROCESSOR FIRST LOAD	
DPPXDBAT	DPPXDBAT		DATA BASE FINAL PHASE PROCESSOR SECOND LOAD	
DPPXDBCP	DPPXDBCP		DATA BASE BDAM DATA SET COMPRESS	
DPPXDBDA	DPPXDBDA		DATA BASE FINAL PHASE PROCESSOR SUPPORT ROUTINE TO WRITE DATA TO DATA BASE BDAM DATA SETS	
DPPXDBIN	DPPXDBIN		OFFLINE DATA BASE TABLE CONSTRUCT	
		DPXDBIN1	OBTAIN INFORMATION FOR DB TABLES	
		DPXDBIN2	INITIALIZE DB TABLES	
		DPXDBIN3	SORT VS RESIDENT ARRAYS BY USE CODE	
		DPXDBIN4	WRITE OUT DB TABLES	
		DPXDBIN6	READ PDS DIRECTORY FOR DB	
DPPXDBLG	DPPXDBLG		DATA BASE FINAL PHASE PROCESSOR SUPPORT ROUTINE TO CALCULATE LOGGING ARRAY BLOCK COUNT AND BLOCK SIZE	

CSECT NAME	MAIN SEGMENT	COPIED SEGMENT	
DPPXDEFL	DPPXDEFL		DEFINE LOCK CONTROL BLOCKS
DPPXDPB	DPPXDPB		DATA RECORDING PLAYBACK
DPPXDRC	DPPXDRC		DATA RECORDING COLLECTION ROUTINE
DPPXDRCX	DPPXDRCX		DUMMY DATA RECORDING COLLECTION
DPPXIMPP	DPPXIMPP		INPUT MESSAGE PROCESSOR
DPPXIMPW	DPPXIMPW		INPUT MESSAGE PROCESSOR WTOR ROUTINE
DPPXKILL	DPPXKILL		ORDERLY TERMINATION ROUTINE
DPPXLOCK	DPPXLOCK		LOCK ROUTINE
DPPXNRTI	DPPXNRTI		DATA PLAYBACK OFFLINE ENTRY ROUTINE
DPPXPCON	DPPXPCON		PLAYBACK REQUEST INTERPRETTER
DPPXRDR	DPPXRDR		DATA PLAYBACK PRINT ROUTINE
DPPXRINT	DPPXRINT		DATA RECORDING INITIALIZATION
DPPXRPRT	DPPXRPRT		REPORT DATA OUTPUT PROCESSOR
DPPXSVCP	DPPXSVCP		SETPSW TYPE 1 SVC
DPPXS2SC	DPPXS2SC		LOCATE INSERT CARDS IN OS/VS1 STAGE II SYSGEN DECK.
DPPXUTIL	DPPXUTIL		OFFLINE UTILITY CONTROL PROGRAM
DPPZSAMP	DPPZSAMP		SAMPLE PROGRAM

MODULE NAMES/HIPO-PDL CHARTS

This section contains a cross-reference of the Special Real Time Operating System module names described by HIPO charts in Section 2 and the corresponding PDL charts in Section 3. COPY Segments are not included. The module names are listed alphabetically for quick reference.

Module Name	HIPO Chart	PDL Chart
DOMIRBT	2-146	3-2
DOMIRCMN	2-148	3-3
DOMIRCPY	2-145	3-4
DOMIRFLV	2-142	3-5
DOMIRFL2	2-143	3-6
DOMIRNIP	2-147	3-8
DOMIRPRB	2-149	3-9
DOMIRWT	2-144	3-10
DOMXSTG1	2-163	3-14
DPPCALCF	2-39	3-45
DPPCPTIM	2-33	3-46
DPPCTIME	2-32	3-47
DPPCTSVC	2-34	3-48
DPPCUPCF	2-40	3-49
DPPDARAY	2-51	3-50
DPPDBLOK	2-48	3-51
DPPDBSIF	2-57	3-52
DPPDFREQ	2-55	3-53
DPPDGETL	2-52	3-54
DPPDITEM	2-49	3-55
DPPDPUTL	2-53	3-56
DPPDSUB2	2-56	3-58
DPPDUMPL	2-54	3-59
DPPDUPDL	2-47	3-60
DPPDWRST	2-151	3-61
DPPFAONC	2-135	3-62
DPPIDBAS	2-45	3-64
DPPIIRB	2-150	3-65
DPPILOGN	2-46	3-66
DPPINIT	2-9	3-67
DPPINITO	2-11	3-68
DPPINIT1	2-12	3-69
DPPIPFIX	2-113	3-70
DPPIPFRE	2-114	3-71
DPPISTAE	2-13	3-72
DPPITIMI	2-31	3-73
DPPMINIT	2-59	3-74
DPPMMSG	2-60	3-75
DPPMMSGV	2-62	3-76
DPPMMSG1	2-61	3-77
DPPPARM	2-139	3-78
DPPPIF	2-121	3-79
DPPSAMP1	2-166	3-80
DPPSASOC	2-94	3-81
DPPSBFST	2-100	3-82

Module Name	HIPO Chart	PDL Chart
DPPSBF1	2-85	3-83
DPPSCHCK	2-80	3-84
DPPSCHPR	2-95	3-88
DPPSCLUP	2-78	3-89
DPPSCL1	2-86	3-90
DPPSCMPR	2-93	3-91
DPPSCRBK	2-92	3-93
DPPSINIT	2-77	3-96
DPPSLOCK	2-96	3-102
DPPSMSGI	2-89	3-103
DPPSMSGO	2-97	3-104
DPPSNOTE	2-82	3-105
DPPSNTPT	2-81	3-106
DPPSOPCL	2-98	3-107
DPPSOP1	2-87	3-108
DPPSPNTF	2-83	3-110
DPPSRCIO	2-99	3-111
DPPSRDWT	2-84	3-112
DPPSRTCP	2-93.1	3-116.1
DPPSRSTR	2-79	3-116
DPPSSHAR	2-101	3-117
DPPSSRCH	2-102	3-118
DPPSST1	2-88	3-119
DPPSSWCH	2-91	3-120
DPPSTBOS	2-90	3-121
DPPSUNLK	2-103	3-122
DPPSUNSH	2-104	3-123
DPPSWRST	2-105	3-124
DPPSXTCB	2-106	3-125
DPPTCBGT	2-112	3-126
DPPTCSVC	2-111	3-127
DPPTDLMP	2-26	3-128
DPPTDSVC	2-22	3-129
DPPTETXR	2-19	3-130
DPPTGWFW	2-108	3-131
DPPTIMPS	2-27	3-132
DPPTPMON	2-16	3-133
DPPTPSVC	2-21	3-134
DPPTPWQE	2-24	3-135
DPPTQIMP	2-28.1	3-135.1
DPPTRGWA	2-110.1	3-136
DPPTRSVC	2-23	3-137
DPPTSMON	2-20	3-138
DPPTSTAE	2-28	3-139
DPPTWQDL	2-25	3-140
DPPTWSVC	2-110	3-141
DPPUMSG	2-159	3-142

Module Name	HIPO Chart	PDL Chart
DPPXDBAS	2-155	3-145
DPPXDBAT	2-156	3-146
DPPXDBCP	2-161	3-147
DPPXDBDA	2-157	3-148
DPPXDBIN	2-160	3-149
DPPXDBLG	2-158	3-150
DPPXDEFL	2-115	3-151
DPPXDPB	2-74	3-152
DPPXDRC	2-71	3-153
DPPXDRCX	2-72	3-154
DPPXIMPP	2-65	3-155
DPPXIMPW	2-64	3-156
DPPXKILL	2-66	3-157
DPPXLOCK	2-116	3-158
DPPXNRTI	2-162	3-159
DPPXPCON	2-73	3-160
DPPXRDR	2-74.1	3-161
DPPXRINT	2-70	3-162
DPPXSVCP	2-68	3-163
DPPXRPRT	2-117	3-164
DPPXUTIL	2-154	3-165
DPPZSAMP	2-165	3-166

MACRO NAMES/MODULE NAMES

This section contains a cross-reference of the Special Real Time Operating System Macro names to the name of the module that receives control when the corresponding macro is executed. The Macro names are listed alphabetically for quick reference.

Macro Name	Module Name	HIPO Chart	PDL Chart
CBGET	DPPTCBGT	2-112	3-126
CBFREE	DPPTCBGT	2-112	3-126
CHAIN	DPPTCSVC	2-111	3-127
DDSBLDL	DPPSBF1	2-85	3-83
DDSCLOSE	DPPSCL1	2-86	3-90
DDSFIND	DPPSBF1	2-85	3-83
DDSOPEN	DPPSOP1	2-87	3-108
DDSSTOW	DPPSST1	2-88	3-119
DEFLOCK	DPPXDEFL	2-115	3-151
DPATCH	DPPTDSVC	2-22	3-129
DPPFIX	DPPIPFIX	2-113	3-70
DPPFREE	DPPPIPFRE	2-114	3-71
DUMPLOG	DPPDUMPL	2-54	3-59

Macro Name	Module Name	HIPO Chart	PDL Chart
FREEWA	DPPTGWFW	2-108	3-131
GETARRAY	DPPDARAY	2-51	3-50
GETBLOCK	DPPDBLOK	2-48	3-51
GETITEM	DPPDITEM	2-49	3-55
GETLOG	DPPDGETL	2-52	3-54
GETWA	DPPTGWFW	2-108	3-131
LOCK	DPPXLOCK	2-116	3-158
MESSAGE	DPPMMSG	2-60	3-75
PATCH	DPPTPSVC	2-21	3-134
PTIME	DPPCTSVC	2-34	3-48
PURGEWQ	DPPTPWQE	2-24	3-135
PUTARRAY	DPPDARAY	2-51	3-50
PUTBLOCK	DPPDBLOK	2-48	3-51
PUTITEM	DPPDITEM	2-49	3-55
PUTLOG	DPPDPUTL	2-53	3-56
RECORD	DPPXDRC	2-71	3-153
REPATCH	DPPTRSVC	2-23	3-137
SETPSW	DPPXSVCP	2-117	3-164
WTFAILDS	DOMIRFLV	2-142	3-5

OPERATOR COMMAND/MODULE NAMES

This section contains a cross-reference of the operator commands recognized by the Special Real Time Operating System to the name of the module that receives control in response to the corresponding operator command. The operator commands are listed alphabetically for quick reference.

Operator Command	Module Name	HIPO Chart	PDL Chart
CANCEL	DPPXKILL	2-66	3-157
DDSCNTRL	DPPSMSGI	2-89	3-103
DLMP	DPPTDLMP	2-26	3-128
DREC	DPPXRINT	2-70	3-162
MSGRC	DPPMMSGV	2-62	3-76
RTCOPY	DPPSRTCP	2-93.1	3-116.1
REPORT	DPPXRPRT	2-68	3-163
QS	DPPTQIMP	2-28.7	3-135.1
STAE	DPPTIMPS	2-27	3-132
STOP	DPPXIMPW	2-64	3-156

MODULE NAME/FUNCTIONAL AREA

This section contains a cross-reference of the Special Real Time Operating System module names described by HIPO charts in Section 2 and the functional area to which they belong. An overall understanding of the functional area of concern can be obtained in the Description and Operations Manual. The module names are listed alphabetically for quick reference.

Module Name	Functional Area
DOMIRBT	Two CPU Operation
DOMIRCMN	Two CPU Operation
DOMIRCPY	Two CPU Operation
DOMIRFLV	Two CPU Operation
DOMIRFL2	Two CPU Operation
DOMIRNIP	Two CPU Operation
DOMIRPRB	Two CPU Operation
DOMIRWT	Two CPU Operation
DOMXSTG1	SYSGEN Utility
DPPCALCF	Time Management
DPPCPTIM	Time Management
DPPCTIME	Time Management
DPPCTSVC	Time Management
DPPCUPCF	Time Management
DPPDARAY	Data Base Management
DPPDBLOK	Data Base Management
DPPDBSIF	Data Base Management
DPPDFREQ	Data Base Management
DPPDGETL	Data Base Management
DPPDITEM	Data Base Management
DPPDPUTL	Data Base Management
DPPDSUB2	Data Base Management
DPPDUMPL	Data Base Management
DPPDUPDL	Data Base Management
DPPDWRST	Two CPU Operation
DPPFAONC	High-Level Language Support
DPPIDBAS	Data Base Management
DPPIIRB	Two-CPU Operation
DPPILOGN	Data Base Management
DPPINIT	Initialization
DPPINITO	Initialization
DPPINIT1	Initialization
DPPIPFIX	Supplementary Services
DPPIPFRE	Supplementary Services
DPPISTAE	Initialization
DPPITIMI	Time Management
DPPMINIT	Message Handler

Module Name	Functional Area
DPPMMSG	Message Handler
DPPMMSGV	Message Handler
DPPMMSG1	Message Handler
DPPPARM	High-Level Language Support
DPPPIF	High-Level Language Support
DPPSAMP1	Sample Programs
DPPSASOC	Duplicate Data Set Support
DPPSBFST	Duplicate Data Set Support
DPPSBF1	Duplicate Data Set Support
DPPSCHCK	Duplicate Data Set Support
DPPSCHPR	Duplicate Data Set Support
DPPSCLUP	Duplicate Data Set Support
DPPSCH1	Duplicate Data Set Support
DPPSCMPR	Duplicate Data Set Support
DPPSCRBK	Duplicate Data Set Support
DPPSINIT	Duplicate Data Set Support
DPPSLOCK	Duplicate Data Set Support
DPPSMSGI	Duplicate Data Set Support
DPPSMSGO	Duplicate Data Set Support
DPPSNOTE	Duplicate Data Set Support
DPPSNTPT	Duplicate Data Set Support
DPPSOPCL	Duplicate Data Set Support
DPPSOP1	Duplicate Data Set Support
DPPSPNTF	Duplicate Data Set Support
DPPSRCIO	Duplicate Data Set Support
DPPSRDWT	Duplicate Data Set Support
DPPSRSTR	Duplicate Data Set Support
DPPSRTCP	Duplicate Data Set Support
DPPSSHAR	Duplicate Data Set Support
DPPSSRCH	Duplicate Data Set Support
DPPSST1	Duplicate Data Set Support
DPPSSWCH	Duplicate Data Set Support
DPPSTBOS	Duplicate Data Set Support
DPPSUNLK	Duplicate Data Set Support
DPPSUNSH	Duplicate Data Set Support
DPPSWRST	Duplicate Data Set Support
DPPSXTCB	Duplicate Data Set Support
DPPTCBGT	Supplementary Services
DPPTCSVC	Supplementary Services
DPPTDLMP	Task Management
DPPTDSVC	Task Management
DPPTETXR	Task Management
DPPTGWFW	Supplementary Services
DPPTIMPS	Task Management
DPPTPMON	Task Management
DPPTPSVC	Task Management

Module Name	Functional Area
DPPTPWQE	Task Management
DPPTQIMP	Task Management
DPPTRGWA	Supplementary Services
DPPTRSVC	Task Management
DPPTSMON	Task Management
DPPTSTAE	Task Management
DPPTWQDL	Task Management
DPPTWSVC	Supplementary Services
DPPUMSG	Offline Utility
DPPXDBAS	Offline Utility
DPPXDBAT	Offline Utility
DPPXDBCP	Data Base Compress
DPPXDBDA	Offline Utility
DPPXDBIN	Offline Utility
DPPXDBLG	Offline Utility
DPPXDEFL	Supplementary Services
DPPXDPB	Data Record and Playback
DPPXDRC	Data Record and Playback
DPPXDRCX	Data Record and Playback
DPPXIMPP	Input Message Processing
DPPXIMPW	Input Message Processing
DPPXKILL	Input Message Processing
DPPXLOCK	Supplementary Services
DPPXNRTI	Playback Routine (Offline)
DPPXPCON	Data Record and Playback
DPPXRDR	Data Record and Playback
DPPXRINT	Data Record and Playback
DPPXRPRT	Report Data Output
DPPXSVCP	Supplementary Services
DPPXUTIL	Offline Utility
DPPZSAMP	Sample Program

Appendix B. STORAGE ALLOCATION

PROGRAM STORAGE REQUIREMENTS

Table B-1 shows the approximate Virtual Storage required by the load modules that comprise the Special Real Time Operating System. The total size represents the approximate maximum number of bytes of storage required for all load modules of each function. Several options within these functions or total functions, are selectable at Special Real Time Operating System SYSGEN which may reduce the total size of any SYSGENed system from these values. The table includes estimates for routines that are used in an offline environment only and will never be a part of the online system. The table also includes routines that may be a part of the online system during initialization for a short duration, when requested by the user or while processing unusual conditions.

The frequently used column represents the approximate number of bytes of each function that may be expected to be used frequently in most systems during a continuing realtime execution. The actual use of any function is dependent upon the application programs and, as such, the amount of virtual or real storage occupied by any function is predictable only through analysis of the application.

In addition to the storage represented in Table B-1, approximately 320 bytes are added to the OS/VSI fixed nucleus and 7700 bytes are added to the pageable nucleus.

The Special Real Time Operating System programs also require approximately five cylinders of a 3330 direct access storage device (or equivalent).

These figures do not include the virtual storage or direct access storage that is required for the user's data base. The Special Real Time Operating System requires a minimum data base of approximately 800 bytes of virtual storage.

CONTROL BLOCK STORAGE REQUIREMENTS

In addition to the program and data base storage, the Special Real Time Operating System requires a minimum of approximately 2500 bytes of control block storage. This amount will increase in multiples of 2K as specified by the user. There are a minimum of 5 permanent tasks (job step task plus four subtasks) as part of the Special Real Time Operating System execution. OS/VS1 control block storage (TCBs, RBs and etc.) are not included in the preceding estimates.

Table B-1 Storage Requirements

Function	Frequently Used	Total Size
Task Management	5000	11,000
Time Management	3000	5,000
Data Base	4000	5,000
Data Base Logging		6,000
Message Handler	3300	3,300
Data Recording		7,000
Report Data Output		900
Duplicate Data Set Support	5000	22,000
Input Message Processing		7,400
System Initialization		41,000
Failover/Restart	1000	20,000
FORTRAN PL/I Interface		2,000
Offline Utility Routines		35,000

Appendix C. DATA AREAS

This appendix contains information on the following:

- Overview of Data Areas
- Control Blocks

OVERVIEW OF DATA AREAS

The operation of the Special Real Time Operating System is dependent upon various control block structures. These control blocks are unique to the Special Real Time Operating System job step in which they exist or, if the two-partition operation is invoked, an interface will exist between the control block of the MASTER and SLAVE job steps (partitions). All of these control blocks may be located through a direct pointer in, or a chain of pointers, at the Special Real Time Operating System's Communication Vector Table (SCVT) and subsystem's Communication Vector Table (XCVT). An alternate method of locating the control block chains is through the TCB extension (TCBX).

The address of the TCBX for each Special Real Time Operating System task is contained in the user field of the TCB for that task (see Figure C-1). The control blocks represented in the first portion of Appendix C are not necessarily complete mappings of the control blocks, but are intended to show the logical relationships between the various Special Real Time Operating System data areas to aid in locating the control block required. The second portion of Appendix C contains a complete detailed mapping of each control block and the macro call required to generate a DSECT of that control block. All control blocks will be referenced by DSECT name.

The major control blocks pointers found in the SCVT are shown in Figure C-2 and the functional area associated with each.

The major control blocks referenced by Task Management routines can be located from the Task Management Control Table (TMCT) as shown in Figure C-3. The control blocks related to a Special Real Time Operating System dependent task is demonstrated in Figure C-4 with a non-reentrant module, Y, currently active. The control blocks related to a Special Real Time Operating System independent task is demonstrated in Figure C-5 with a reentrant module, Z, currently active.

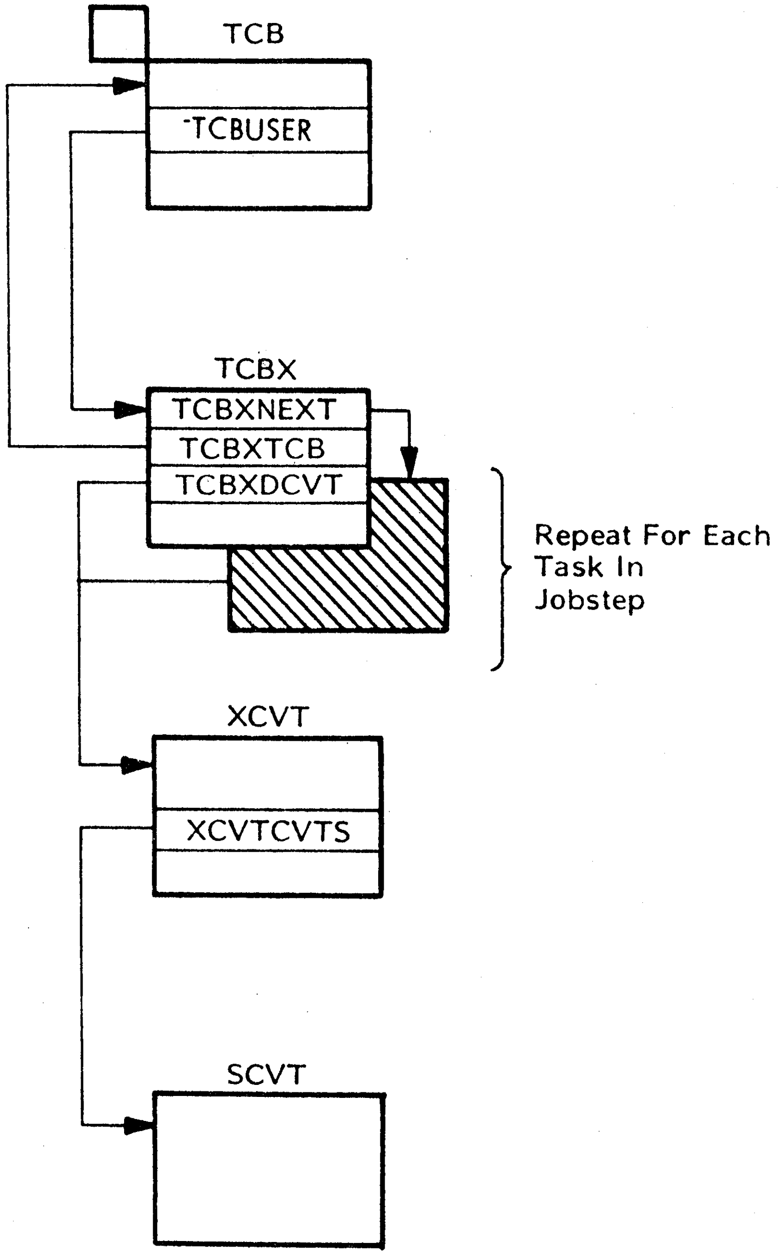


Figure C-1. Special Real Time Operating System Communications Vector Tables

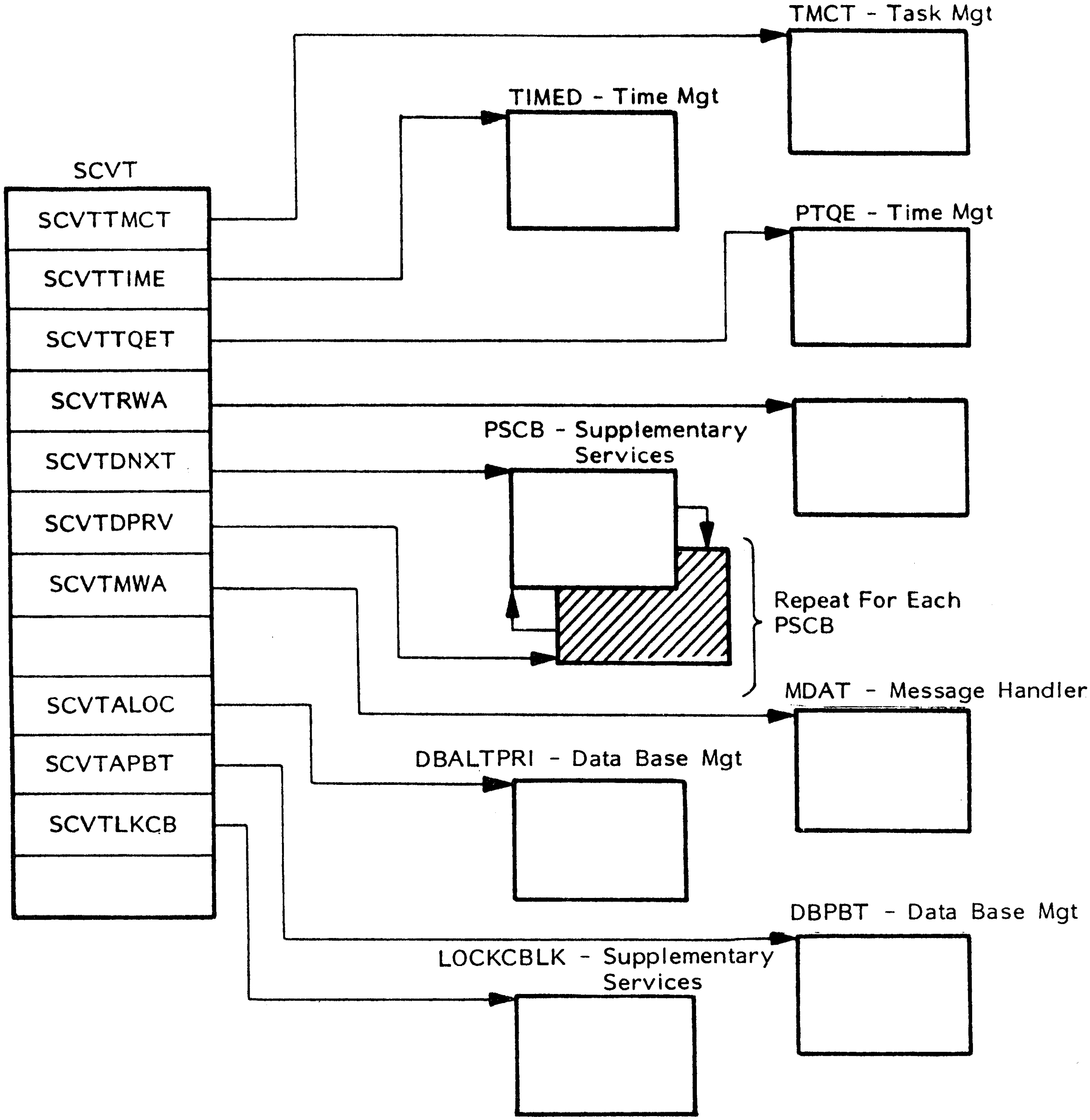


Figure C-2. SCVT Control Blocks

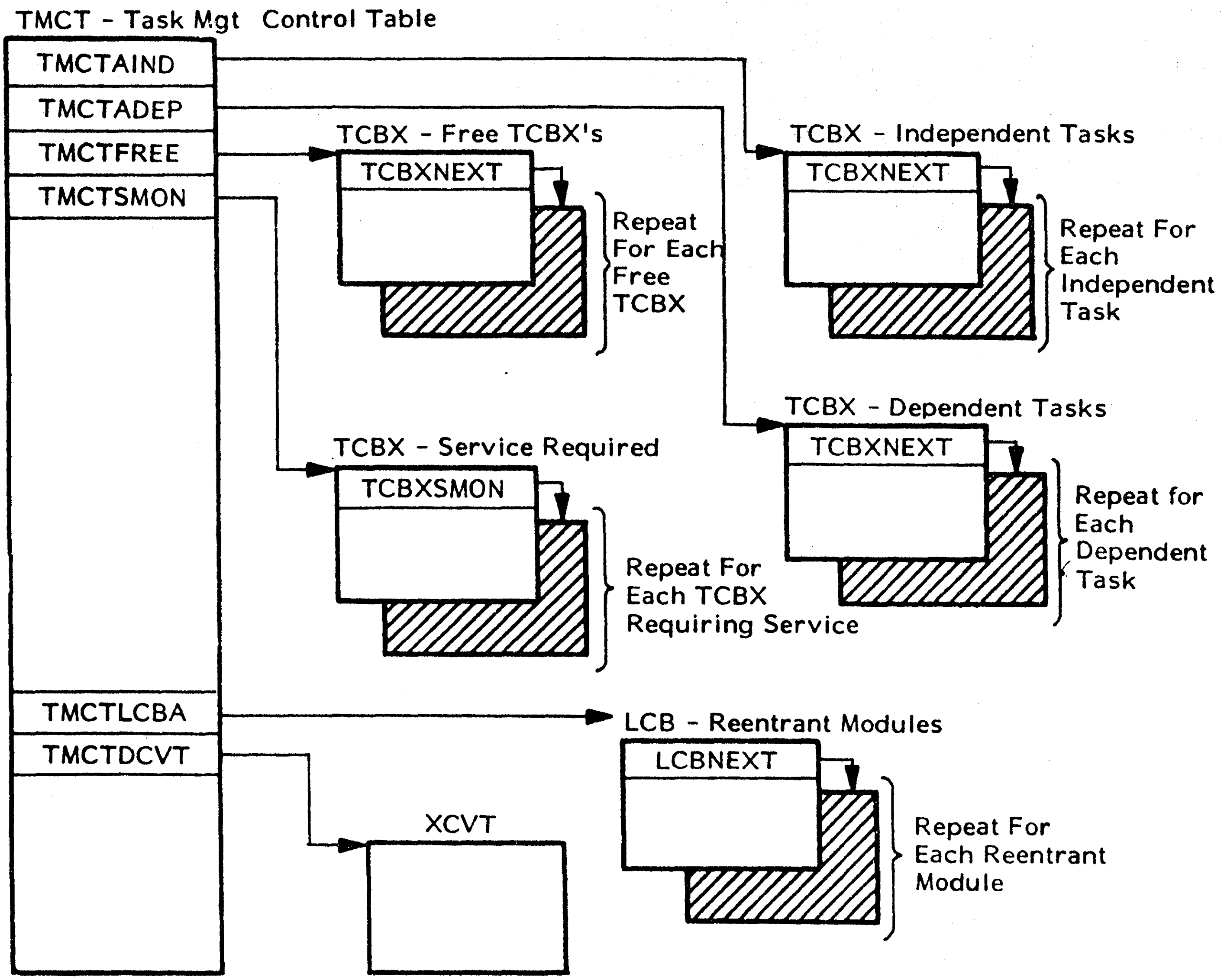


Figure C-3. Task Management Control Tables

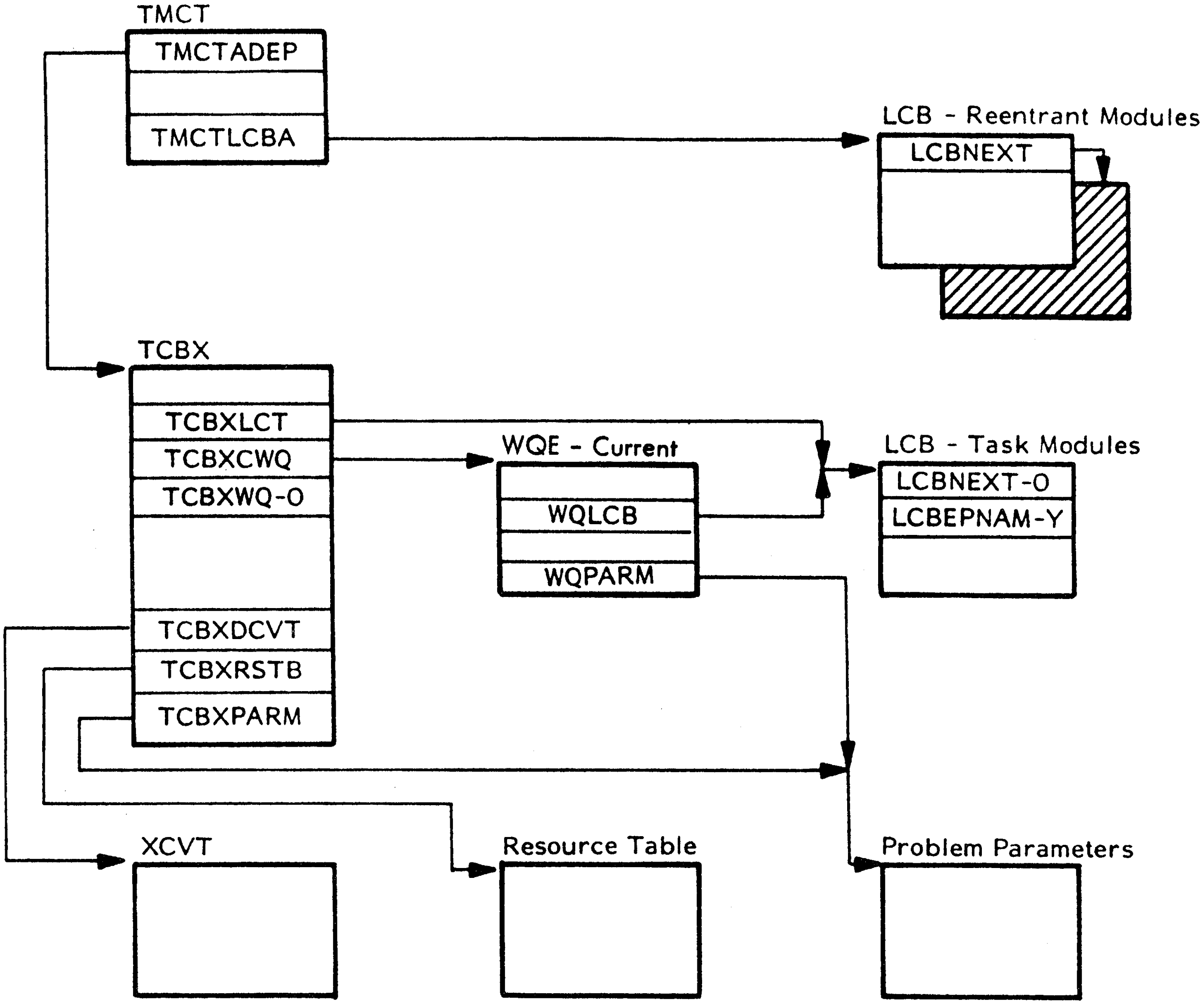


Figure C-4. Control Block Structure For Dependent Task (Dependent Task/Non-reentrant Module Y Currently Active)

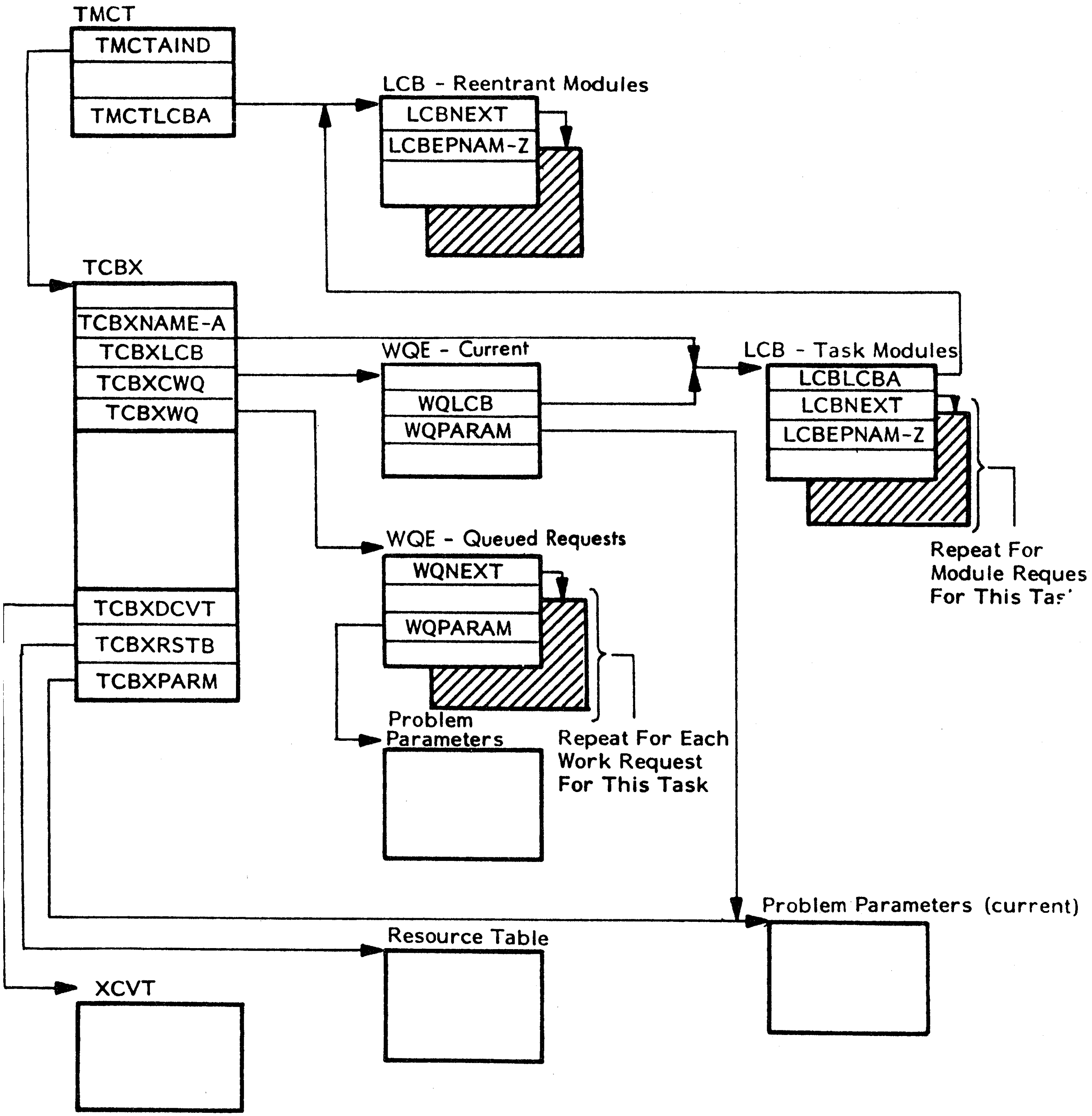


Figure C-5.1. Control Block Structure For QP/QH Tasks (Independent Task A/Reentrant Module Z Currently Active)

The execution of user programs during realtime operation may be monitored by its Generalized Trace Facility (see OS/VS1 Service Aids Manual for description and use of GTF) with a TRACE option of USR specified provided two branch no-op switches in the PATCH monitor routine (DPPTMON) at location DPPTGTF1 and DPPTGTF2 are changed from a X'4700' to a X'45F0'. Patch's to high level language module may be monitored by modifying two other branch no-op switches in DPPTPMON at location DPPTGTF3 and DPPTGTF4 from a X'4700' to a X'45F0'. Abnormal termination (ABEND) of user routines may be monitored by modifying a branch no-op switch in the exit routine, DPPTETXR, at location DPPTGTFS from a X'4700' to a X'45F0'. The user data recorded by GTF for each event will have the following:

GTDSECT

+0	Entry Point name of the user routine	
+8	Task name	
+16	Queue Holder name (if applicable)	
+24	Work Queue ID	A(Active TCB)
+28	GTF ID	A(PATCHing Task TCB)
+32	unused	
+36	unused	
+40	unused	

Figure C-5.2. GTDSECT-User Data Record where GTFID identifies the type of data. That is:

X'01' - indicates entry to user routine.
X'02' - indicates exit from user routine.
X'11' - indicates entry to HLL user routine.
X'12' - indicates exit from HLL use routine.
X'22' - indicates abnormal termination of user routine.

The major control blocks referenced by Data Base Management routines can be located from the Primary Array Locator Table (DBALTPRI) as shown in Figure C-6. The Primary Array Locator Table, as a whole, contains one entry for each data base array. Each entry is described by the DBALTPRI DSECT. These entries are arranged sequentially so that a particular array entry can be referenced by multiplying the length of each array by the array number and then adding the result to the address of origin of the Primary Array Locator Table. The Secondary Array Locator Table, as a whole, contains one entry for each data base array. Each entry is described by the DBALTSEC DSECT These entries are also arranged sequentially for indexing by array number as with the Primary Array Locator Table. The logging control block, as a whole, contains one entry for each log array. Each entry is described by the DBLOGCB DSECT. These entries are arranged sequentially, but are not indexed by the use of an array number. Instead, the associated DBALTPRI for the specified log array contains a numeric value (DBALTNDX) to be added to the address of origin of the log control block to locate the associated DBLOGCB entry. The Direct Access DDNAME table, as a whole, contains one entry for each direct access data set. Each entry is described by the DBDADD DSECT. These entries are also arranged sequentially for indexing by the numeric value (DBALTNDX) as with the log control block. Figure C-7 shows this relationship.

The relationship between the VS resident loggable array and the direct access resident log array is shown in Figure C-8.

The data base offline utility builds the Primary and Secondary Array Locator Tables, the log control block, and the Direct Access DDNAME Table for use in online execution. The data base array (or PDS member), @INIT, contains this information in the data record as shown in Figure C-9. The data base offline utility also constructs a PDS member for each specified array. Figures c-10, C-11, and C-12 show the PDS members build for VS resident arrays, VS resident loggable arrays, and DA resident arrays, respectively.

The two major Time Management control blocks are shown in Figure C-13. The time array described by TIMED DSECT is part of the VS resident data base. The PTOEs are built in response to a PTIME request and are deleted whenever the service request has been completed.

The LOCK/DEFLOCK control blocks (LOCKCBLK and WAITCBLK) are shown in Figure C-14. The LOCKCBLKs are constructed in response to a DEFLOCK request. WAITCBLKs are constructed in the user's save area, whenever a resource is requested by a LOCK macro call but has been previously reserved by another task.

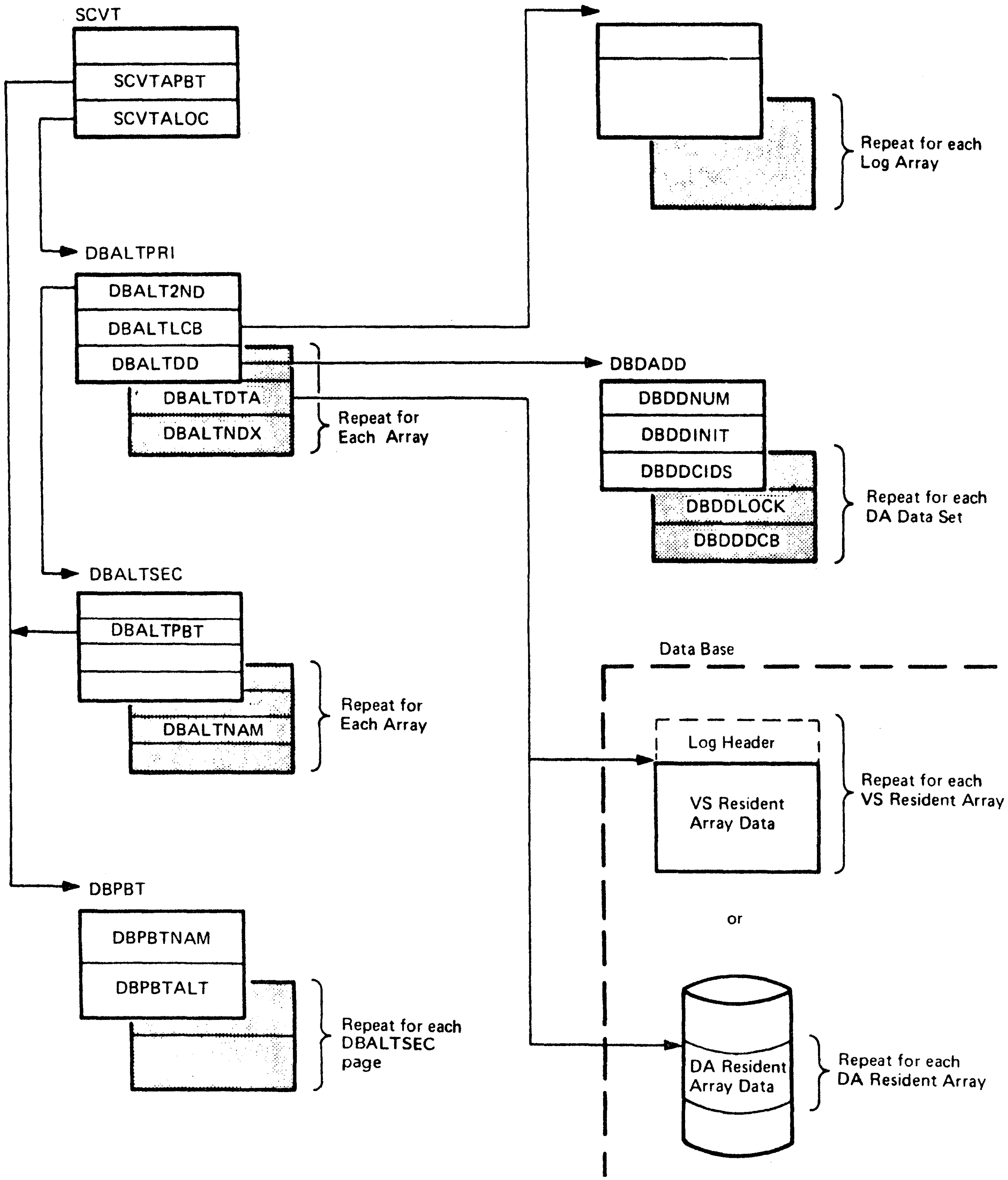


Figure C-6. Data Base Control Block Map

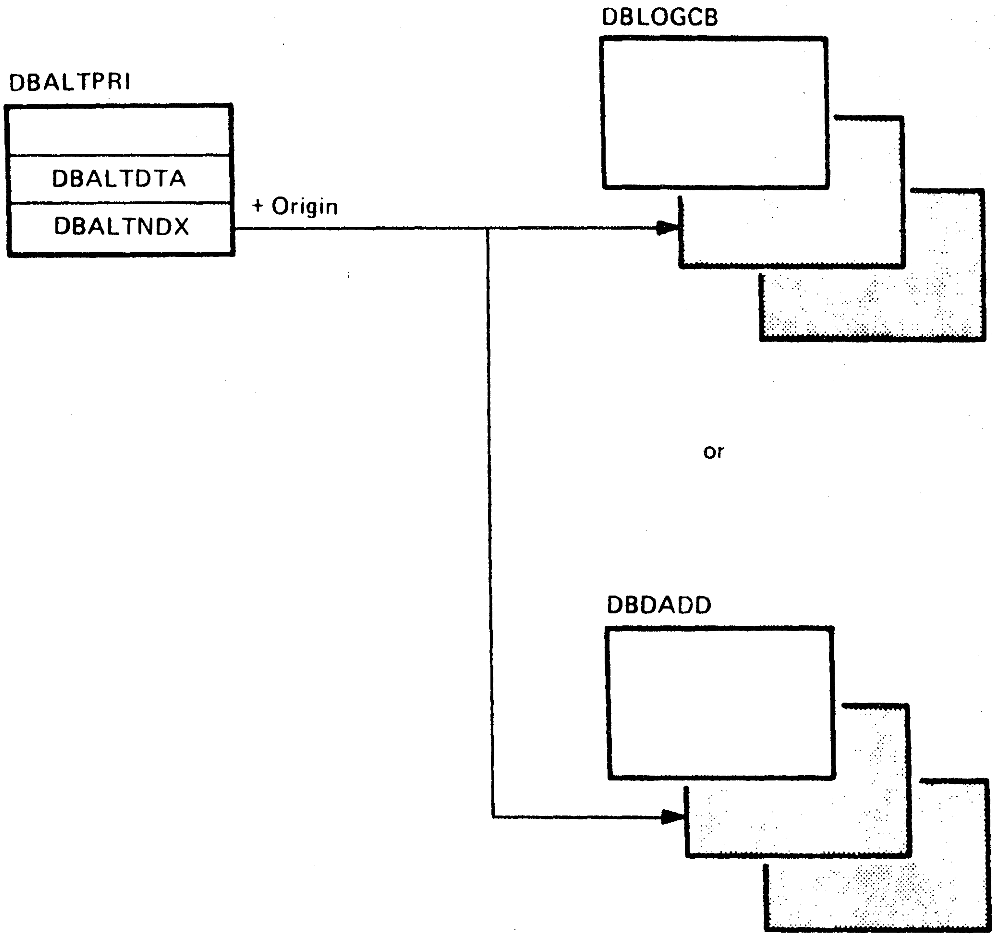


Figure C-7. DBALTPRI Index Pointer for Log Arrays and DA Arrays

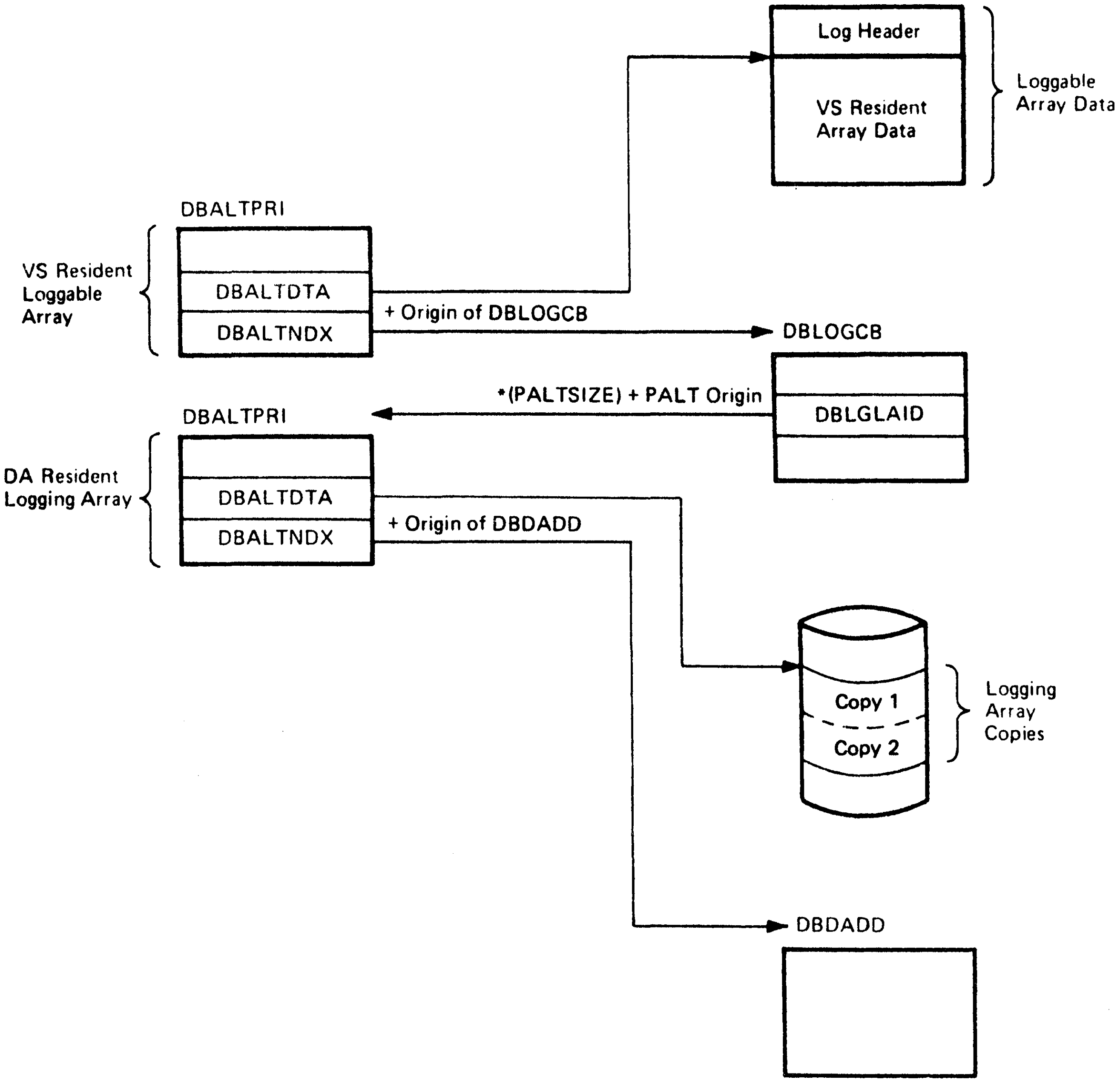


Figure C-8. VS Resident Loggable Array/DA Resident Log Array Relationship

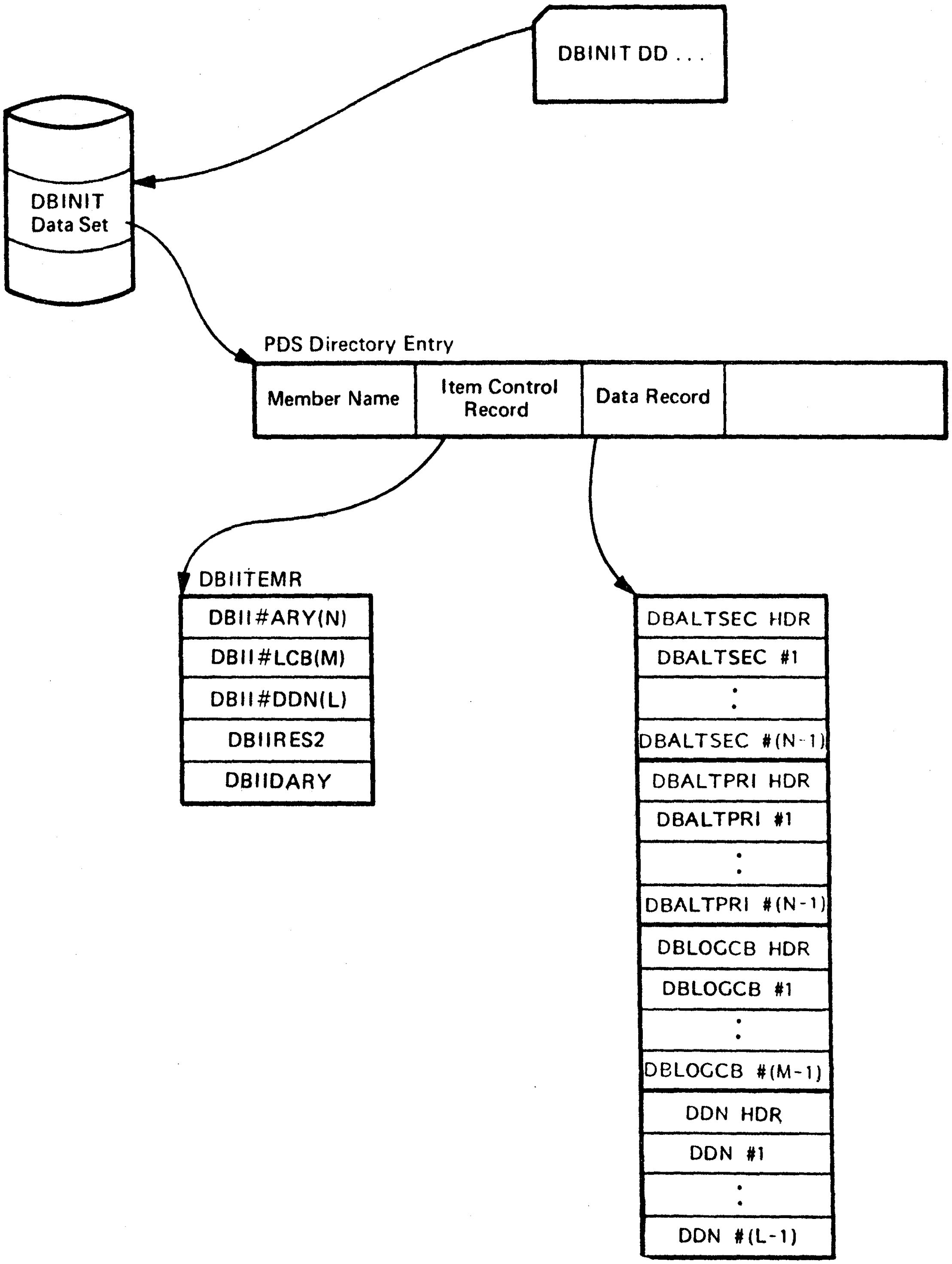


Figure C-9. DB Initialization Array PDS Directory Entry

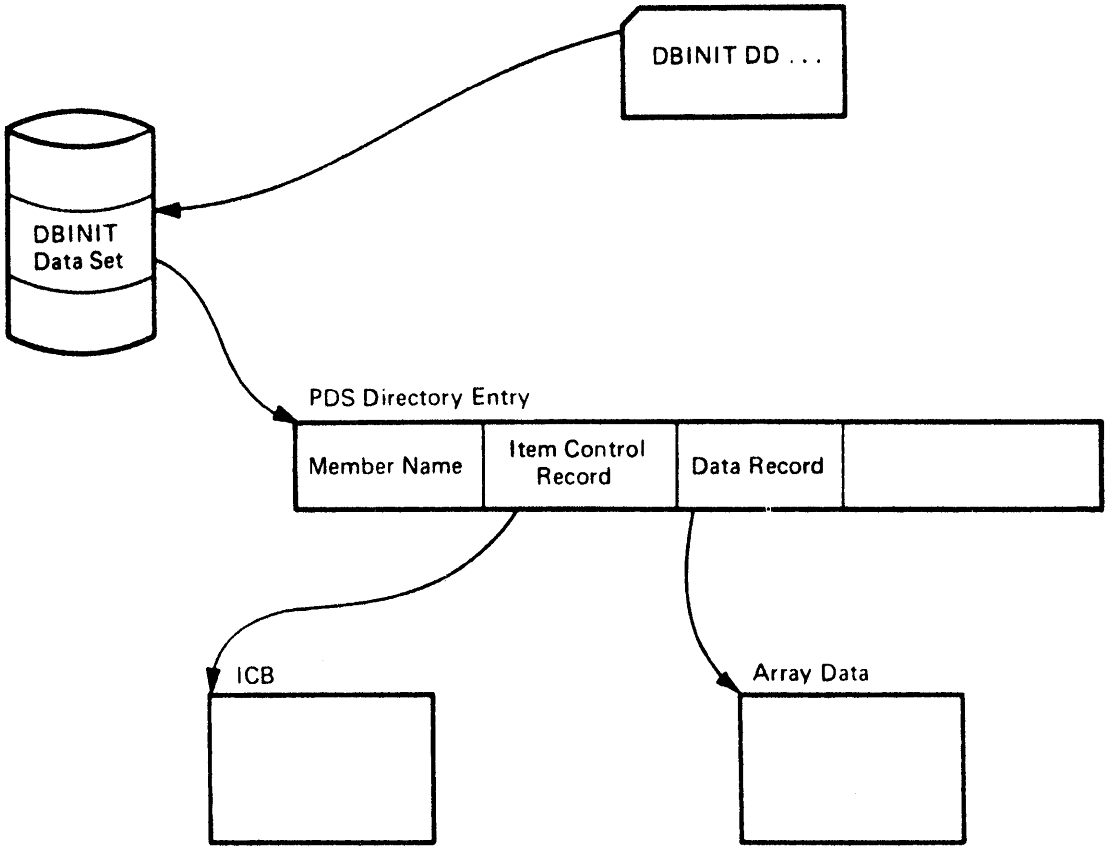


Figure C-10. VS Resident Array PDS Directory Entry

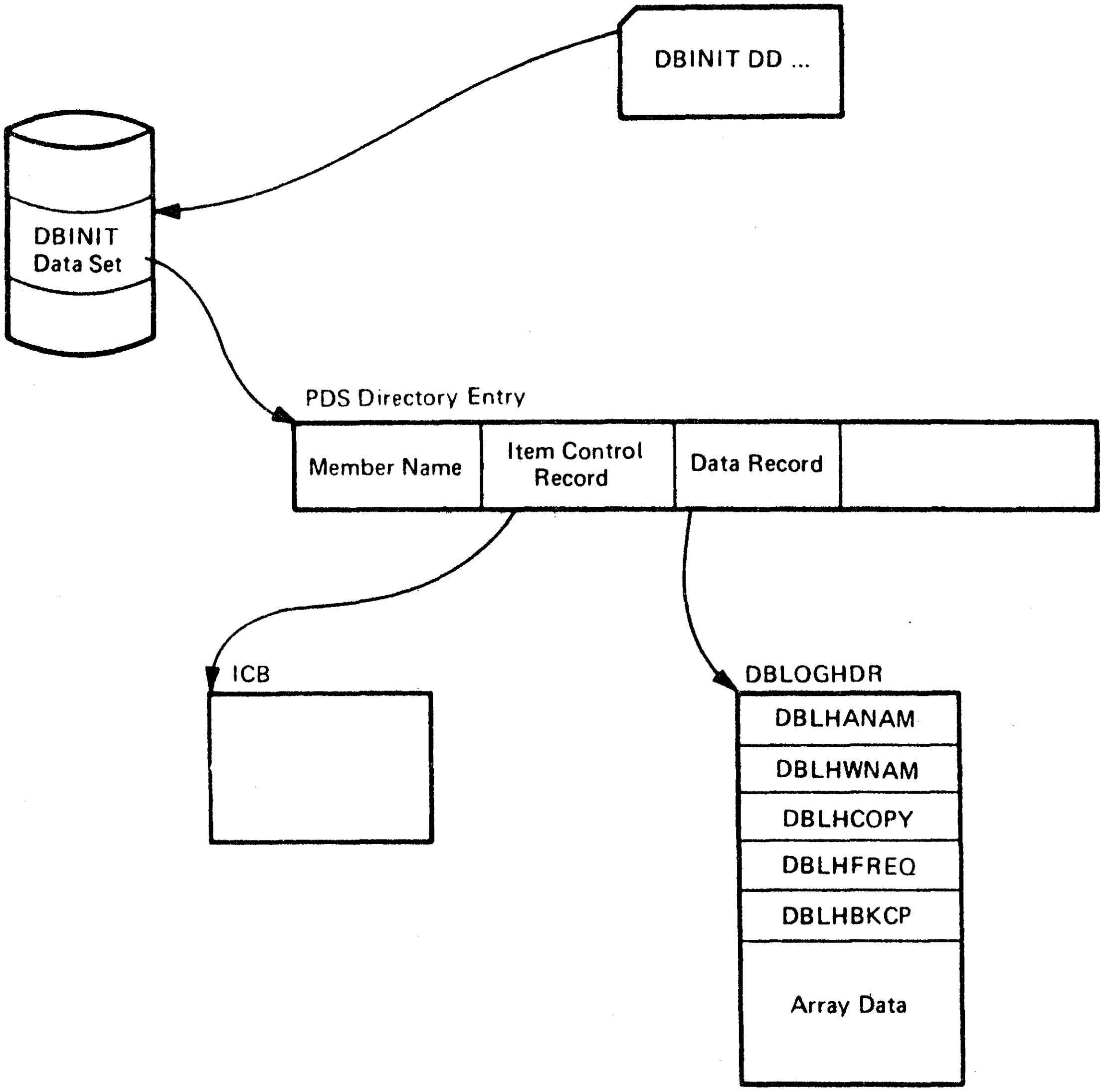


Figure C-11. VS Resident Array with Logging PDS Directory Entry

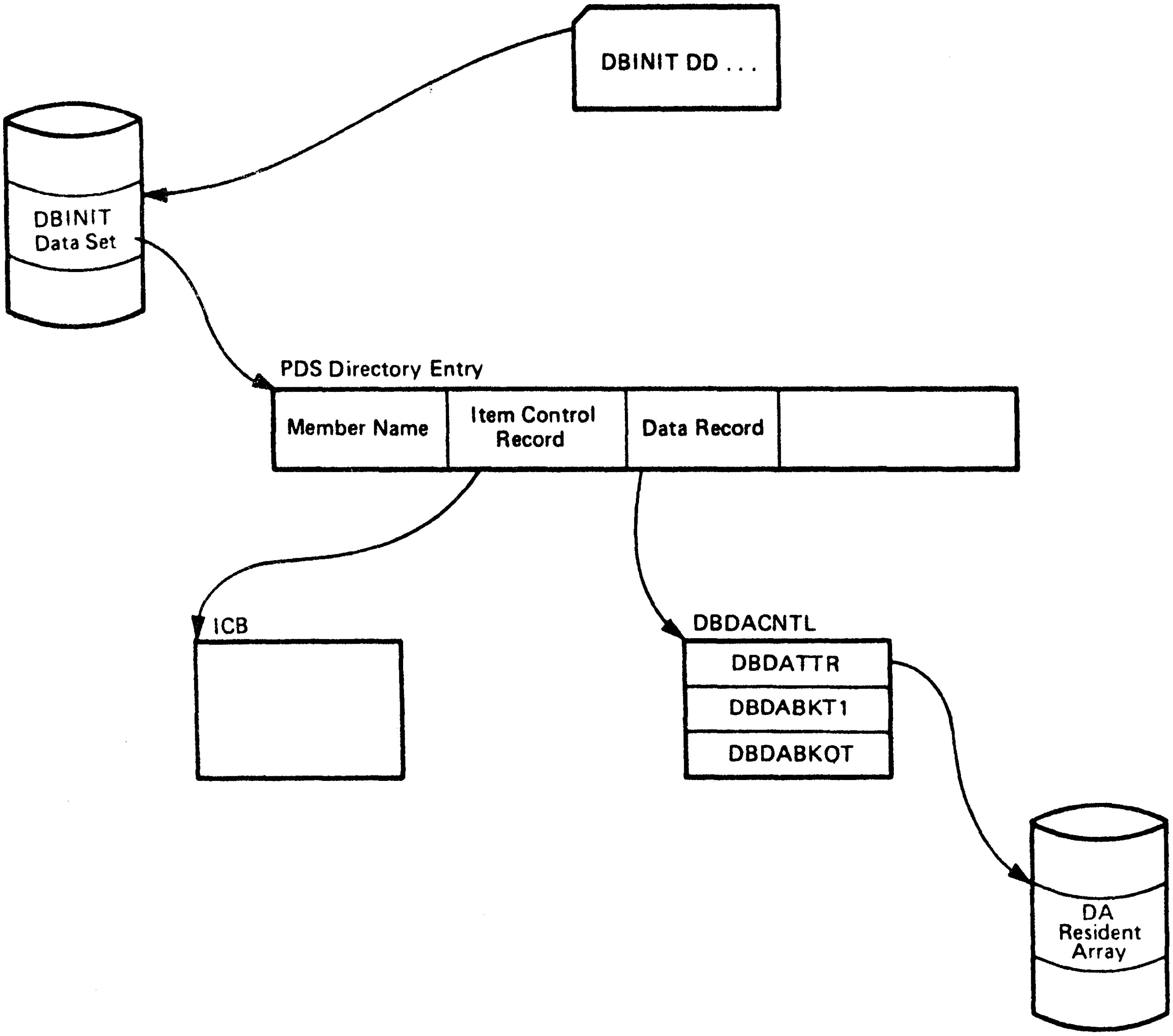


Figure C-12. DA Resident Array PDS Directory Entry

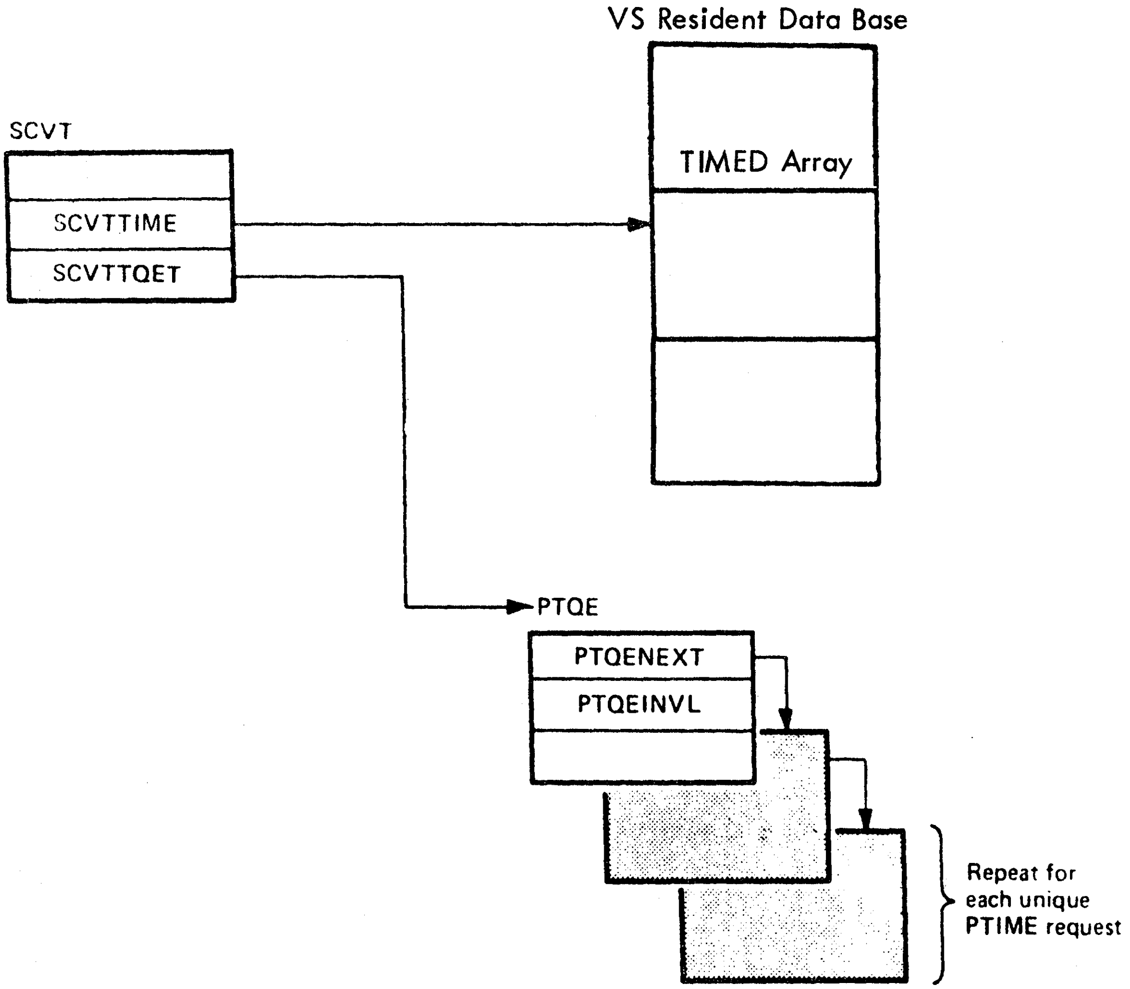


Figure C-13. Time Management Control Blocks

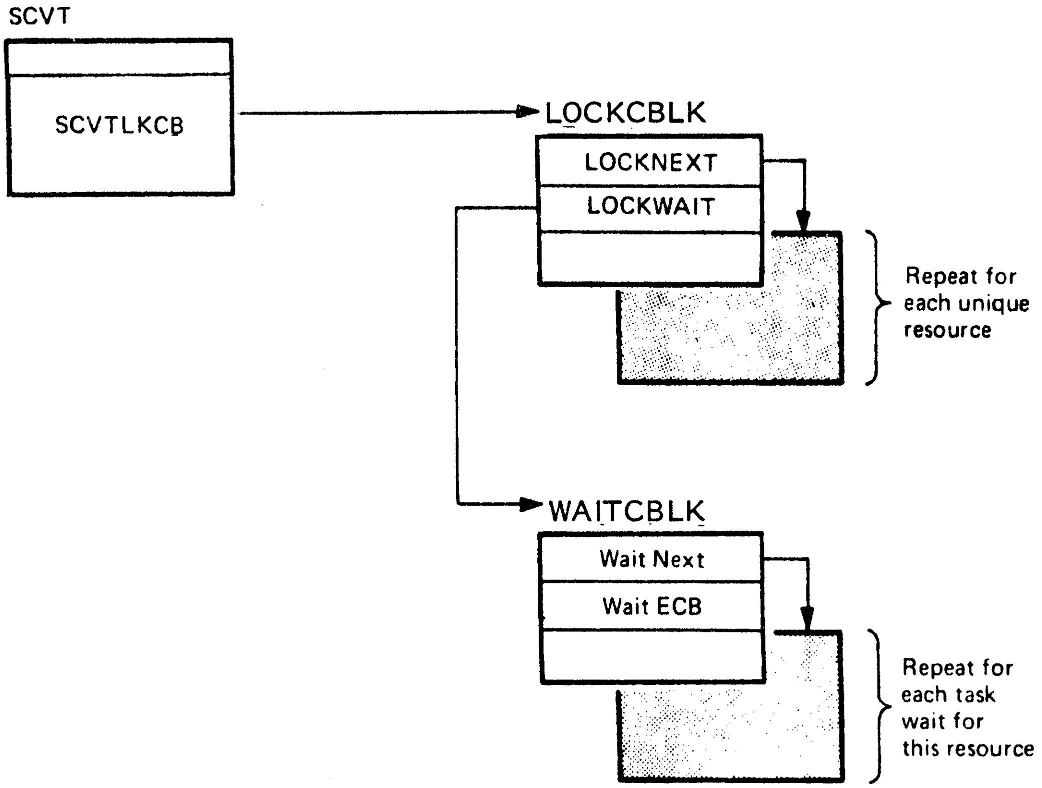

SCVT
SCVTLKCB
LOCKCBLK
LOCKNEXT
LOCKWAIT
Repeat for each unique resource
WAITCBLK
Wait Next
Wait ECB
Repeat for each task wait for this resource

Figure C-14. LOCK-DEFLOCK Control Blocks

The data recording and playback data set is a QSAM sequential data set which contains the data collected as a result of the RECORD macro. It is made up of fixed length physical records which contain variable length logical records. The size of each physical record is specified through the BLKSIZE=parameter of the DRECOUT DD card when recording the data. Any given logical record may span two or more physical records.

The data set consists of three types of logical records: date records, pad records, and data records.

A date record is written at the beginning of the data set and whenever the date in the Special Real Time Operating System Time Array changes.

A pad record is written at the end of a physical record whenever there are less than 50 but more than 0 unused bytes in a physical record.

A data record is written as the result of a RECORD macro.

Whenever a date or data record spans more than one physical record, the data that is not on the initial physical record is preceded by a subheader on each physical record until the end of the logical record.

The format of each logical record is described in Figures C-15 through C-17.

0	1	2	4	5	8	10
flag	unused	length	flag		length	header

10	38	40	42	44
time array	ID	ID	ID	...

Bytes	Field Name	Field Description, Meaning, Contents
4 bits	flag	Flag 4 is specified.
2	length	The entire length of the data.
4 bits	flag	0 - Date record contained entirely within one physical record. 1 - Date record spans two or more physical records.
2	length	Length is 0 if flag is 0. Length is the number of bytes of the logical record contained on the physical record if flag is 1.
28	time array	A copy of the Special Real Time Operating System time array contained in the data base.
2-92	ID, ID, ID,...	Valid three-digit hexadecimal numbers that may be used in recording data.

Figure C-15. Date Record

0	1	End of physical record
Flag	Unused	

Bytes	Field Name	Field Description, Meaning, Contents
4 bits	flag	This field contains a flag of 6 which indicates that the remaining data on this physical record is invalid.

Figure C-16 Pad Record

0		2	4	8	10
flag	ID	length	time	length	header

10	End
Recorded	Data

Bytes	Field Name	Field Description, Meaning, Contents
4 bits	flag	0 - Data contained entirely within one physical record. 1 - Data spans two or more physical records.
12 bits	ID	Three-digit hexadecimal ID supplied during data recording (RECORD macro).
2	length	The length of the entire data record.
4	time	Time contained in the Special Real Time Operating System time array when the data was recorded, measured in decimal 10 milliseconds units - HHMMSSth.
2	length	If flag is 0, length is 0. Length is the number of bytes of the logical record contained in the physical record if flag is 1.

Figure C-17 Data Record

The message data set is a partitioned data set. The data set contains variable length records, the maximum size is 291 bytes. Each message is identified by a unique three-digit number. To avoid duplication of numbers, each subsystem of the Special Real Time Operating System will be assigned a unique set of numbers(this is by convention only) for its message.

1. Directory Entry:

0	8	12	14
DPPnnnb̸b̸	TTRC	Size	

nnn - Message number
b̸b̸ - Blank spaces
TTRC - Relative track position
Size - Message length

2. Member:

0	1	3	4	5	6	10	291
Flag	RC	AC	Lgth	#V	L, CONV, SN, DISP	L	Text

There will be a L, CONV, SN, DISP field for each variable specified.

Flag - Flag byte in hexadecimal
0 - do not affix date to message
1 - affix date to message
RC - routing code in hexadecimal
AC - action code in character
Lgth - length of message in hexadecimal

Figure C-18 Message Data Set Format

The CINFD table is placed in the OS/VS1 nucleus during the Special Real Time Operating System SYSGEN. In systems without External Interrupt Handling, it resides in the pageable nucleus; otherwise, in the fixed nucleus. The SETPSW routine returns the address of this table in register 1.

CINFD

+0	Contain the constant C'CINF'	
+4	CINFDTIT Address of the Type I SVC Branch Table in module DOMISVC1 (DPPTYP1T)	
+8	CINFDT2T Address of the Type II SVC Branch Table in module DOMISVC2 (DPPTYP2T)	
+12	CINFDPRB Nonzero if continuous monitor or PROBE active	CINFDFRC Failover/ restart writes can be done if zero.

Figure C-19 CINFD Without External Interrupt Handling

The remainder of the CINFD table exists only in systems with external interrupt handling.

$+12_{10}$		CINFDPSW		
$+16_{10}$	CINFDPSW Save OS/VS1 EXT.NEW PSW			
$+20_{10}$	CINFDPSW	CINFDTPE External time standard period in seconds	RESERVED	
$+24_{10}$	CINFDPSS Address of substitute external FLIH (DOMICEXT)			
$+28_{10}$	CINFDEB1 ECB for Failover confirmed external interrupt			
$+32_{10}$	CINFDTC1 Addr of TCB that is waiting on CINFDEB1			
$+36_{10}$	CINFDEB2 ECB for External Time sync interrupt			
$+40_{10}$	CINFDTC2 Addr of TCB that is waiting on CINFDEB2			
$+44_{10}$	CINFDCLK TOD clock value when external time sync received			
$+48_{10}$	CINFDCLK			
$+52_{10}$	POST Eligibility Bits (Note 1)	CINFDFLI X'00'- Substitute Ext.FLCH not initialized	CINFDCC Condition code from STCK instruction (Bits 2-3 of Byte)	CINFDGEN Feature Bytes (Note 2)

Figure C-20 CINFD with Externai Interrupt Handling

The remainder of the CINFD table exists only in systems with Special Real Time Operating System clock comparator support.

	CINFDEB3
$+56_{10}$	ECB for clock comparator interrupt
	CINFDTC3
$+60_{10}$	Addr of TCB that will wait on above

Note 1

CINFDEC1	1xxx xxxx	CINFDEB1 can be posted
CINFDEC2	x1xx xxxx	CINFDEB2 can be posted
CINFDEC3	xx1x xxxx	CINFDEB3 can be posted

Note 2

CINFDBTM	1xxx xxxx	Ext. time standard SYSGENed
CINFDBSW	x1xx xxxx	Failover confirmed interrupt SYSGENed
CINFDBCP	xx1x xxxx	Special Real Time Operating System clock comparator support SYSGENed
CINFDEC4	xxx1 xxxx	Set when failure confirmed signal received

Figure C-21 CINFD with Clock Comparator Support

This section describes the first 620 bytes of the failover/restart bootstrap module (Module DOMIRBT).

+0	DOMBVAD1 Virtual address of first 2K of bootstrap
+4	DOMBRAD1 Real address of first 2K of bootstrap
+8	DOMBVAD2 Virtual address of second 2K of bootstrap
$+12_{10}$	DOMBRAD2 Real address of second 2K of bootstrap
$+16_{10}$	DOMBVAD3 Virtual address of DOMIRNIP
$+20_{10}$	DOMBRAD3 Real address of DOMIRNIP
$+24_{10}$	DOMBVWK Virtual address of work area
$+28_{10}$	DOMBRWK (64 bytes) Up to 16 real addresses of work area (32K maximum)
$+92_{10}$	DOMBWKL Length of work area (32K maximum) (Note 1)
$+96_{10}$	DOMBCHRS CCHH of start of real storage dump on failover/restart

Figure C-22 DOMBOOTH --Failover/Restart Bootstrap Header (Page 1 of 6)

$+100_{10}$	DOMBCHRE CCHH of end of Real Storage dump on failover/restart data set	
$+104_{10}$	DOMBCHPS CCHH of start of paging data sets copy on failover/ restart data set	
$+108_{10}$	DOMBCHPE CCHH of end of paging data sets copy on failover/ restart data set	
$+112_{10}$	DOMBCHJQ CCHH of start of SYS1.SYSJOBQE dump on failover/ restart data set	
$+116_{10}$	DOMBCHJQ CCHH of end of SYS1.SYSJOBQE dump or failover/ restart data set	
$+120_{10}$	DOMBC2JQ CCHH of start of first SYS1.SYSJOBQE data set	
$+124_{10}$	DOMBC2JQ CCHH of end of first SYS1.SYSJOBQE data set	
$+128_{10}$	DOMBC3JQ Addr of UCB containing first SYS1.SYSJOBQE data set.	DOMBCHJ2 (162 Bytes) CCHHCCHH/ CCHHCCHH & UCB for next 9 SYS1.SYSJOBQE data sets if any
$+292_{10}$	DOMBCHSW CCHH of start of SYS1.SYSWADS dump on failover/restart data set	
$+296_{10}$	DOMBSCHSW CCHH of end of SYS1.SYSWADS dump on failover/restart data set	

Figure C-22 DOMBOOTH--Failover/Restart Bootstrap Header (Page 2 of 6)

<table>
<tr><td>$+300_{10}$</td><td colspan="2">DOMBC2SW
CCHH of start of SYS1.SYSWADS data set</td></tr>
<tr><td>$+304_{10}$</td><td colspan="2">DOMBC2SW
CCHH of end of SYS1.SYSWADS data set</td></tr>
<tr><td>$+308_{10}$</td><td>DOMBC3SW
UCB addr for SYS1.SYSWADS</td><td>DOMBCHS1
CCHH of start of SWADS dump on failover/restart</td></tr>
<tr><td>$+312_{10}$</td><td>Data set (MASTER partition)</td><td>DOMBCHS1
CCHH of end of SWADS dumper on failover/restart</td></tr>
<tr><td>$+316_{10}$</td><td>Data set (MASTER partition)</td><td>DOMBC2S1
CCHH of start of SWADS1 data set</td></tr>
<tr><td>$+320_{10}$</td><td>(MASTER partition)</td><td>DOMBC2S1
CCHH of end of SWADS1 data set</td></tr>
<tr><td>$+324_{10}$</td><td></td><td>DOMBC3S1
UCB address of SWADS data set</td></tr>
<tr><td>$+328_{10}$</td><td colspan="2">DOMBCHS2
CCHH of start of SWADS2 dump on failover/restart data set (SLAVE Partition)</td></tr>
<tr><td>$+332_{10}$</td><td colspan="2">DOMBCHS2
CCHH of end of SWADS2 dump on failover/restart data set (SLAVE Partition)</td></tr>
<tr><td>$+336_{10}$</td><td colspan="2">DOMBC2S2
CCHH of start of SWADS2 data set (SLAVE Partition)</td></tr>
<tr><td>$+340_{10}$</td><td colspan="2">DOMBC2S2
CCHH of end of SWADS2 data set (SLAVE Partition)</td></tr>
<tr><td>$+344_{10}$</td><td>DOMBC3S2
UCB address SWADS2 data set</td><td>DOMBCHSP
Reserved</td></tr>
</table>

Figure C-22 DOMBOOTH--Failover/Restart Bootstrap Header (Page 3 of 6)

$+348_{10}$	Reserved	
$+352_{10}$	Reserved	Reserved
$+356_{10}$	DOMBTCBU Address of MASTER RT Job Step TCB	
$+360_{10}$	DOMBTCBO Address of SLAVE RT Job Step TCB or zero	
$+364_{10}$	DOMBWTOA Address of a 3210 or 3215 console	Reserved
$+368_{10}$	DOMBCTLR Control registers 0-15 (64 bytes)	
$+432_{10}$	DOMBGPR Registers 0-15 for reserve (64 bytes)	
$+496_{10}$	DOMBRPSW Resume PSW (8 bytes)	
$+500_{10}$	DOMBRPSW	
$+504_{10}$	DOMBCPUI Target for a STIDP instruction	
$+508_{10}$	DOMBCPUI	
$+512_{10}$	DOMBCKC Clock comparator value at restart write	

Figure C-22 DOMBOOTH--Failover/Restart Bootstrap Header (Page 4 of 6)

$+516_{10}$	DOMBCKC	
$+520_{10}$	DOMBCK TOD clock value at restart write	
$+524_{10}$	DOMBCK	
$+528_{10}$	DOMBPT CPU timer at restart write	
$+532_{10}$	DOMBPT	
$+536_{10}$	DOMBSTE Real storage size	
$+540_{10}$	DOMBTYP UCBTYP field of failover/restart data set	
$+544_{10}$	DOMBMAXB Maximum blocksize of device containing failover/ restart data set	
$+548_{10}$	DOMBCC Number cyl of device containing failover/ restart data set	DOMBHH Tracks/cyl of device containing failover/restart data set
$+552_{10}$	DOMBFACT Device dependent information	
$+556_{10}$	DOMBTOL Device dependent information	

Figure C-22 DOMBOOTH--Failover/Restart Bootstrap Header (Page 5 of 6)

$+560_{10}$	DOMBDSCB DSNAME (44 bytes) and BBCCHHK of format 1 DSCB of failover/restart data set (52 bytes)
$+612_{10}$	DOMBTIME Registers 0 and 1 from TIME macro at restart write
$+616_{10}$	DOMBTIME

Note 1: The length of the work area is the same as the maximum blocksize of the device where the failover/restart data set is allocated.

Figure C-22 DOMBOOTH--Failover/Restart Bootstrap Header (Page 6 of 6)

CONTROL BLOCKS

The following pages contain reference information about content and format of Special Real Time Control Blocks. The control blocks described here are used by more than one module of the system. The block descriptions appear in this section in the order that they appear in this index.

DSECT	DESCRIPTION	MACRO CALL
DBALTPRI	Primary Array Locator Table	DBALTPRI DSECT
DBALTSEC	Secondary Array Locator Table	DBALTSEC DSECT
DBARRAYD	Array Macro Expansion	DBARRAYD DSECT
DBBLOCKD	Block Macro Expansion	DBBLOCKD DSECT
DBDACNTL	DA Array Control Header	DBDACNTL DSECT
DBDADD	DA DDNAME Table	DBDADD DSECT
DBDEFD	DBDEF Macro Expansion	DBDEFD DSECT
DBDIRB	Data Base Directory Entry - BLDL	DBDIRB DSECT
DBDIRR	Data Base Directory Entry - READ	DBDIRR DSECT
DBDMPHDR	DUMPLOG Header Record	DBDMPHDR DSECT
DBITEMD	Item Macro Expansion	DBITEMD DSECT
DBLOGCB	Data Base Logging Control Block	DBLOGCB DSECT
DBLOGHDR	Data Base Logging Header	DBLOGHDR DSECT
DBPBT	Data Base Page Boundary Table	DBPBT DSECT
WAREA	Data Base Offline Work Area	DBWAREA
BRT1	Type 1 SVC Branch Table	DPPXBRT
BRT2	Type 2 SVC Branch Table	DPPXBRT
DRT	Data Recording Table	DRECBLKS
GFCB	GETWA/FREEWA Control Block	DPPXBLKS GFCB=Y
GFBE	GETWA/FREEWA Block Entry	DPPXBLKS GFCB=Y
IMP	Input Message Processing Table	IMPBLKS
LCB	Load Control Block	DPPXBLKS LCB=Y
LOCKCBLK	LOCK Control Block	DPPXBLKS LOCK=Y
WAITCBLK	WAIT Control Block	UPPXBLKS LOCK=Y
RCT	Message Pointing Code Table	MSGBLKS
MDT	Message DCB Table	MSGBLKS
MDAT	Message Address Table	MSGBLKS
PTIMEL	PTIME Parameter List	DPPXBLKS PTIMEL=Y
PTQE	PTIME Queue Element	DPPXBLKS PTQE=Y
PWQE	PURGEWQ Parameter List	PWQE
REPL/SUPL	PATCH/REPATCH Parameter List	DPPXBLKS REPL=Y
PROBL	User Parameter List	DPPXBLKS REPL=Y
SCVT	Special Real Time Operating System Communications Vector Table	DPPXBLKS SCVT=Y
STAEBLK	STAE Command Control Block	STAEBLK
STAEXBK	STAE User EXIT Interface	STAEXBK
TCBX	TCB Extension	DPPXBLKS TCBX=Y
TIMED	Time Array DSECT	DPPXBLKS TIMED=PTIME
TMCT	Task Management Control Table	DPPXBLKS TMCT=Y
WQE	Work Queue Element	DPPXBLKS WQ=Y
XCVT	Subsystem Communication Vector Table	DPPXBLKS XCVT=Y

Figure C-23 (1 Of 2). DBALTPRI

```
    ***********************************************************************
    *                                                                     *
    *                PRIMARY ARRAY LOCATOR TABLE                          *
    *                                                                     *
    ***********************************************************************
    *                                                                     *
    *  FUNCTION - CONTAIN HIGH USAGE INFORMATION FOR LOCATING AND USING   *
    *             A DATA BASE ARRAY.                                      *
    *                                                                     *
    *  REFERENCED BY -  CONTROL BLOCK = SCVT     - LABEL = SCVTALOC       *
    *                                                                     *
    ***********************************************************************
000 DBALTPRI DSECT
    ***********************************************************************
    *                                                                     *
    *  THESE LABELS DESCRIBE THE DATA IN THE FIRST ENTRY (ENTRY 0) OF     *
    *  THE TABLE.  ENTRY 0 CONTAINS CONTROL INFORMATION AND POINTERS.     *
    *                                                                     *
    ***********************************************************************
000 DBALTBG1 DS    0F
000 DBALT2ND DS    F                    A(SECONDARY ALT)
004 DBALTLCB DS    F                    A(LOGGING CONTROL BLOCK TABLE)
008 DBALTDC  DS    F                    A(DA DDNAME TABLE)
00C DBALTEN1 DS    0F
    DBALTSZ1 EQU   DBALTEN1-DBALTBG1   PRIMARY ALT ENTRY SIZE
    ***********************************************************************
    *                                                                     *
    *  THESE LABELS DESCRIBE THE DATA IN ALL SUBSEQUENT ENTRIES OF        *
    *  THE TABLE.  THERE IS ONE ENTRY FOR EACH DATA BASE ARRAY.           *
    *                                                                     *
    ***********************************************************************
00C          ORG   DBALTBG1
    *DBRES    BIT  7          ON - DA RESIDENT --- OFF - VS RESIDENT
000 DBRES    DS    XL(B'00000001')
001          ORG   *-B'00000001'
    *DBBLOCK  BIT  6          ON - BLOCKED      --- OFF - UNBLOCKED
000 DBBLOCK  DS    XL(B'00000010')
002          ORG   *-B'00000010'
    *DBALIGN  BIT  5          ON - VS PAGE BOUNDARY ORIGIN
    *                         OFF - VS DOUBLEWORD BOUNDARY ORIGIN
000 DBALIGN  DS    XL(B'00000100')
004          ORG   *-B'00000100'
    *DBMIN    BIT  4          ON - ALIGN TO OCCUPY MINIMUM NUMBER OF VS
    *                              PAGES - VALID ONLY IF BIT 5 IS OFF
    *                         OFF - USE BIT 5 TO DETERMINE BOUNDARY
000 DBMIN    DS    XL(B'00001000')
008          ORG   *-B'00001000'
    *DBINIT   BIT  3          ON - INITIALIZE ARRAY DATA TO VS - VALID
    *                              ONLY IF BIT 7 IS OFF
    *                         OFF - DO NOT INITIALIZE ARRAY DATA TO VS
000 DBINIT   DS    XL(B'00010000')
010          ORG   *-B'00010000'
    *DBREINT  BIT  2          ON - AFTER RESTART, REINITIALIZE VS ARRAY
    *                              FROM LOG DATA SET - VALID ONLY IF
    *                              BIT 7 IS OFF
    *                         OFF - DO NOT REINITIALIZE ARRAY
000 DBREINT  DS    XL(B'00100000')
```

Figure C-23 (2 Of 2). DBALTPRI

```
020          ORG   *-B'00100000'
    *DBLOG    BIT  1        ON - LOGABLE VS ARRAY - VALID ONLY IF
    *                            BIT 7 IS OFF
    *                       OFF - ARRAY IS NCT LOGABLE
000 DBLOG    DS    XL(B'01000000')
040          CRG   *-B'0100C000'
    *DBDUMMY  BIT  0        CN - DUMMY ARRAY - NO INITIALIZATION OR
    *                            PROCESSING WILL BE PERFCRMED
    *                       OFF - REAL ARRAY
000 DBDUMMY  DS    XL(B'10000000')
080          ORG   *-B'10000000'
000 DBALTFLG DC    X'00'
    DBALTCTA EQU   DBALTFLG
001          DS    AL3                 DATA ADCRESS/TTR
004 DBALTBCT DS    H                   BLOCK CCUNT
006 DBALTBAS DS    H                   BLOCK SIZE/ARRAY SIZE
008 DBALTBFT DS    XL1                 BLOCKS ON FIRST TRACK
009 DBALTBOT DS    XL1                 BLCCKS CN OTHER TRACKS
00A DBALTNDX DS    H                   DADD/LOGCB TABLE INDEX
```

Figure C-24. DBALTSEC

```
    ***********************************************************************
    *                                                                     *
    *              SECCNDARY ARRAY LOCATOR TABLE                          *
    *                                                                     *
    ***********************************************************************
    *                                                                     *
    *  FUNCTICN - CCNTAIN LOW LSAGE INFORMATICN FOR LOCATINC AND LSING    *
    *             A CATA BASE ARRAY                                       *
    *                                                                     *
    *  REFERENCEC BY -  CCNTROL BLOCK = CBALTPRI - LABEL = CBALT2ND       *
    *                   CONTROL BLOCK = DBPBT    - LABEL = CBPBTALT       *
    ***********************************************************************
000 DBALTSEC DSECT
    ***********************************************************************
    *                                                                     *
    *  THESE  LABELS CESCRIBE TFE CATA IN THE FIRST ENTRY (ENTRY 0) CF    *
    *  THE TABLE.  ENTRY 0 CONTAINS CONTROL INFORMATICN AND POINTERS.     *
    *                                                                     *
    ***********************************************************************
000 CBALTBG2 DS    0F
000 CBALT#NO DS    H                   NUMBER OF NUMBEREC ARRAYS PLLS CNE
002 DBALTNLM DS    H                   NUMBERED OF ARRAYS PLUS ONE
004 DPALTPBT DS    F                   A(PAGE BOUNCARY TABLE
008 DBALTPBS DS    F                   SIZE OF PAGE BOUNCARY TABLE
00C OBALTUNS DS    F                   UNUSED
010 DBALTEN2 DS    0F
    DBALTSZ2 EQU   CBALTEN2-DBALTBG2   SECCNCARY ALT ENTRY SIZE
    ***********************************************************************
    *                                                                     *
    *  THESE LABELS DESCRIBE THE CATA IN ALL SLBSECUENT ENTRIES OF        *
    *  THE TABLE.  THERE IS ONE ENTRY FOR EACH CATA BASE ARRAY.           *
    *                                                                     *
    ***********************************************************************
010          ORG   DBALTBG2
000 CBALTNAM DS    CL8                 ARRAY NAME
008 DBALTICB DS    F                   ITEM CONTROL BLOCK TTR
00C CBALTIRS DS    H                   NUMBER OF ITEM CCNTRCL BLCCKS
00E DBALTUSE DS    XL1                 ARRAY LSE CODE
00F DBALTLCG DS    XL1                 ARRAY LCG FRECUENCY CODE
    DPALTREC EQU   2048                ITEM CONTROL BLCCK RECORD SIZE
```

Figure C-25. DBARRAYD

```
    ***********************************************************************
    *                                                                     *
    *              ARRAY  MACRC EXPANSION DSECT                           *
    *                                                                     *
    ***********************************************************************
000 DBARRAYD DSECT
000 CBARSTRT DS    0F
000 DBARNAME DS    CL8                 ARRAY NAME
008 DBARBKCT DS    H                   BLOCK CCUNT
00A CBARBKSZ DS    H                   BLOCK SIZE
00C CBARCCNM DS    CL8                 DA/LOG DD NAME
014 DBARES   DS    XL(B'00000001')
015          ORG   *-B'00000001'
014 DBABLOCK DS    XL(B'00000010')
016          ORG   *-B'0000001C'
014 DBAALIGN DS    XL(B'00000100')
018          ORG   *-B'00000100'
014 DBAMIN   DS    XL(B'00001000')
01C          ORG   *-B'00001000'
014 DBAINIT  DS    XL(B'00010000')
024          ORG   *-B'00010000'
014 DBAREINT DS    XL(B'00100000')
034          ORG   *-B'00100000'
014 DBALCG   DS    XL(B'01000000')
054          ORG   *-B'01000000'
014 DBADUMMY DS    XL(B'10000000')
094          ORG   *-B'10000000'
014 DBARFLGS CC    X'00'
015 DBARUSE  DS    X                   USE CODE
016 DBADALCG CS    XL(B'10000000')
096          CRG   *-B'10000000'
016 DBARFLG2 DC    X'00'
017 CBARUNUS DS    X                   UNUSEC
018 CBARUPD  DS    H                   UPCATE LEVEL
01A CBARUNS2 CS    H                   UNUSEC
01C CBARICBS DS    V                   A(ITEM CONTRCL BLCCK START)
020 DBARICBP DS    V                   A(ITEM CCNTRCL BLCCK STOP)
024 CBARCTAS DS    V                   A(ITEM CATA START)
028 DBARDTAP DS    V                   A(ITEM CATA STOP)
02C DBARNOIT DS    A                   NUMBER CF ITEMS
030 CBARNOBK DS    A                   NUMBER OF BLOCK CCNTROL BLCCKS
034 CBARSTOP DS    0F
    CBARSIZE EQU   DBARSTOP-CBARSTRT
```

Figure C-26. DBBLOCKD

```
    ***********************************************************************
    *                                                                     *
    *              BLCCK  MACRC EXPANSION DSECT                           *
    *                                                                     *
    ***********************************************************************
000 CBBLCCKD DSECT
000 CBBKSTRT DS    0F
000 DRBKSNUM DS    H                   START BLOCK NUMBER
002 CBBKPNUM DS    H                   STOP BLOCK NUMBER
004 CPBKBKIS DS    V                   A(BLOCKED ICB START)
008 DBBKBKIP DS    V                   A(PLOCKED ICB END)
00C DPBKBKDS DS    V                   A(BLOCKED ITEM DATA START)
010 DBBKBKCP DS    V                   A(BLOCKED ITEM CATA END)
014 DRBKSTOP DS    0F
    CBBKSIZE FQU   CBBKSTOP-DBBKSTRT
```

Figure C-27. DBDACNTL

```
    ***********************************************************************
    *                                                                     *
    *         DA ARRAY CONTROL HEACER                                     *
    *                                                                     *
    ***********************************************************************
000 DBCACNTL DSECT
000 DPCABGN  DS    0F
000 DBDANAME DS    CL8            DA ARRAY NAME
008 DBDATTR  DS    AL4            RELATIVE TTR CN CA DATA SET
00C CBCABKT1 DS    XL1            RECORDS WRITTEN CN FIRST DA TRACK
00D DBDABKCT DS    XL1            RECORDS WRITTEN CN OTHER DA TRACKS
00E CBCAUNUS DS    XL2            UNUSED
010 DBDAEND  DS    0F
    DPCASIZE EQU   DBCAEND-DBDABGN
010 DBDACTTR DS    XL4                 CBCACNTL RECORD TTR CN PDS
014 DBDABKCT DS    XL2                 CA ARRAY BLCCK COUNT
016 DBDABKSZ DS    XL2                 DA ARRAY BLCCK SIZE
018 DBCASTCP DS    0F
    DBCALNTH EQU   DBDASTOP-DBCABGN
```

Figure C-28 (1 Of 3). DBDADD

```
    ***********************************************************************
    *                                                                     *
    *                DIRECT ACCESS DDNAME TABLE                           *
    *                                                                     *
    ***********************************************************************
    *                                                                     *
    *  FUNCTION - CCNTAINS A DCB FOR EACH CATA BASE CATA SET.  EACH       *
    *             BDAM DCB IS PRECEEDED BY TFE ADDR OF A LOCK CNTL BLOCK  *
    *                                                                     *
    *  REFERENCED BY -  CCNTROL BLOCK = CBALTPRI - LABEL = CBALTDD        *
    *                                                                     *
    ***********************************************************************
000 DBDADD   DSECT
    ***********************************************************************
    *                                                                     *
    *  HEACER FCR DCNAME TABLE                                            *
    *                                                                     *
    ***********************************************************************
000 DBDDBGN  DS    0F
000 DBDDNUM  DS    F                      NUMBER CF ENTRIES
004 DBDDUPD  DS    H                      CURRENT UPCATE LEVEL
006          DS    H                      SPARE
008          DS    F                      SPARE
    ***********************************************************************
    *                                                                     *
    *  DATA BASE PARTITIONED CATA SET CCB                                 *
    *                                                                     *
    ***********************************************************************
    *                            CATA CCNTROL BLOCK
    *
00C DBDDINIT DC    0F'0'                       ORIGIN CN WORD BCUNDARY
    *                            DIRECT ACCESS DEVICE INTERFACE
00C          DC    BL16'0'                      FDAD,DVTBL
01C          DC    A(0)                         KEYLE,DEVT,TRBAL
    *                            COMMCN ACCESS METHOD INTERFACE
020          DC    AL1(0)                      BUFNO
021          DC    AL3(1)                      BUFCB
024          DC    AL2(0)            BUFL
026          DC    BL2'0000001C0C000000'                  DSORG
028          DC    A(1)                          IOBAD
    *                            FOUNCATION EXTENSICN
02C          DC    BL1'00C0000C'                        BFTEK,BFLN,HIARCHY
02D          DC    AL3(1)                      ECDAD
030          DC    BL1'00000000'               RECFM
031          DC    AL3(0)                      EXLST
    *                            FOUNCATION BLOCK
034          DC    CL8'DBINIT'                 DDNAME
03C          DC    BL1'00000010'               OFLGS
03D          DC    BL1'00000000'                            IFLG
03E          DC    BL2'0010010C00100100'       MACR
    *                            BSAM-BPAM-QSAM INTERFACE
040          DC    BL1'00000000'                                       RER1
041          DC    AL3(1)                      CFECK, GERR, PERR
044          DC    A(1)                        SYNAD
048          DC    H'0'                        CIND1, CIND2
04A          DC    AL2(0)                      BLKSIZE
04C          DC    F'0'                        WCPC, WCPL, CFFSR, OFFSW
```

Figure C-28 (2 Of 3). DBDADD

```
050          DC    A(1)                    ICBA
054          DC    AL1(0)                  NCP
055          DC    AL3(1)                  ECBR, EOBAD
    *                          BSAM-BPAM INTERFACE
C58          DC    A(1)                    EOBW
05C          DC    H'0'                    DIRCT
05E          DC    AL2(0)         LRECL
060          DC    A(1)                    CNTRL, NOTE, POINT
C64 DBDDEND1 DS    0F
    DBDDLN1  EQU   CBDDEND1-CBDDINIT
    ***********************************************************************
    *                                                                     *
    *  DCB FOR REACING COMPOSITE ITEMS ARRAY                              *
    *                                                                     *
    ***********************************************************************
    *                         DATA CCNTROL BLOCK
    *
064 DBCDCIDS DC    0F'0'                   ORIGIN CN WORD BOUNCARY
    *                         DIRECT ACCESS DEVICE INTERFACE
064          DC    BL16'0'                  FCAD,DVTBL
074          DC    A(0)                     KEYLE,DEVT,TRBAL
    *                         COMMON ACCESS METHOD INTERFACE
078          DC    AL1(0)                  BUFNC
079          DC    AL3(1)                  BUFCB
07C          DC    AL2(0)              BUFL
07E          DC    BL2'010000CC0C000000'             DSORG
080          DC    A(1)                     IOBAD
    *                         FOUNCATION EXTENSION
084          DC    BL1'000C0000'                 BFTEK,BFLN,HIARCHY
C85          DC    AL3(1)                  EOCAD
088          DC    BL1'00000000'           RECFM
089          DC    AL3(0)                  EXLST
    *                         FCUNDATION BLOCK
08C          DC    CL8'DBCIDS'             DCNAME
094          DC    BL1'00000010'           OFLGS
095          DC    BL1'CC0C0000'                         IFLG
096          DC    3L2'001C000C00100000'   MACR
    *                         BSAM-BPAM-QSAM INTERFACE
098          DC    BL1'00000000'                                    RER1
099          DC    AL3(1)                  CFECK, GERR, PERR
09C          DC    A(1)                    SYNAD
0A0          DC    H'0'                    CIND1, CIND2
0A2          DC    AL2(0)                  BLKSIZE
0A4          DC    F'0'                    WCPO, WCPL, CFFSR, OFFSW
0A8          DC    A(1)                    IOBA
0AC          DC    AL1(0)                  NCP
0AD          DC    AL3(1)                  ECBR, EOBAD
    *                          BSAM-BPAM INTERFACE
0B0          DC    A(1)                    EOBW
0B4          DC    H'0'                    CIRCT
0B6          DC    AL2(0)         LRECL
0B8          DC    A(1)                    CNTRL, NCTE, PCINT
0BC DBDDEND2 DS    0F
    DBDDLN2  EQU   DBDDEND2-DBCDCIDS
0BC DBDDHEND DS    0F
    DBDDHDRS EQU   DBCHEND-CBCDBGN
    *                                                                     *
```

Figure C-28 (3 Of 3). DBDADD

```
    *  END OF HEADER AREA                                                *
    **********************************************************************
0BC          ORG   CBDCBGN
    **********************************************************************
    *                                                                    *
    *  BCAM CATA SET LOCK CONTRCL BLCCK POINTER ANC CCB ENTRIES.        *
    *  ONE ENTRY FOR EACH BDAM DATA SET IN THE DATA BASE.                *
    *                                                                    *
    **********************************************************************
000 DBDDLCCK DS    A                   A(LOCK CONTROL BLCCK FOR THIS DCB)
004          ORG   *+16
    *,***  IHB061  DDNAME NOT SPECIFIEC
    *                       DATA CONTROL BLOCK
    *

014          ORG   *-16                    TC ELIMINATE UNUSED SPACE
004 DBCDDCB  DS    OF'0'                    ORIGIN CN WORD BOUNDARY
004          ORG   *+16                    TO ORIGIN GENERATION
014          DC    A(0)                     KEYLE,DEVT,TRBAL
    *                       COMMON ACCESS METHCD INTERFACE
018          DC    AL1(0)                  BUFNO
019          DC    AL3(1)                  BUFCB
01C          DC    AL2(0)             BUFL
01E          DC    BL2'001C00CC000C0000'             DSORG
020          DC    A(1)                     IOBAD
    *                       FOUNDATION EXTENSION
024          DC    BL1'00000000'                  BFTEK,BFLN,HIARCHY
025          DC    AL3(1)                  ECDAD
028          DC    BL1'00000000'           RECFM
029          DC    AL3(0)                  EXLST
    *                       FOUNCATION BLOCK
02C          DC    CL8'0'                  DDNAME
034          DC    BL1'00000010'           OFLGS
035          DC    BL1'00000000'                          IFLG
036          DC    BL2'00111C0000111010'   MACR
    *                       BDAM INTERFACE
038          DC    BL1'0000000C'
039          DC    AL3(1)                  CHECK
03C          DC    A(1)                    SYNAD
040          DC    H'0'
042          DC    AL2(0)                  BLKSIZE
044          DC    A(1)                    IOBSQ
048          DC    A(1)                    SCND
04C          DC    A(1)                    ICBUQ
050          DC    A(1)                    UCND
054          DC    A(0)                    LIMCT
058          DC    F'1'                    XCNT,XARG
05C          DC    A(1)                    DRDX
060          DC    A(1)                    DFOR
064          DC    A(1)                    CFBK
068          DC    A(1)                    DYNB
06C DBDDEND  DS    OF
    CBDDESZ  EQU   DBDDEND-CBDCLOCK
```

Figure C-29. DBDEFD

```
    *********************************************************************
    *                                                                   *
    *              CBCEF  MACRC EXPANSION DSECT                         *
    *                                                                   *
    *********************************************************************
000 DBDEFD   DSECT
000 DBDFSTRT DS    0F
000 DBDFHDS  DS    0F
000 DBDFNAMA DS    A                   A(FPP NAME)
004 DBDFNAME DS    CL8                 FPP NAME
00C DBDFFPPE DS    V                   A(ENC CF FPP INPLT)
010 DBDFHDP  DS    0F
    DBDFHCSZ EQU   CBDFHDP-CBDFHDS
010          CRG   DBDFHDS
000 DBDFOPTN DS    X                   OPTION BYTE
001 CBDFUNUS DS    XL3                 UNUSED
004 DBDFDEFE DS    V                   A(END OF THIS DBDEF)
008 DBDFNOAR DS    A                   NUMBER OF ARRAYS IN THIS DBDEF
00C DBDFSTOP DS    0F
    DBDFSIZE EQU   DBDFSTOP-DBDFSTRT
```

Figure C-30 (1 Of 2). DBDIRB

```
    ************************************************************************
    *                                                                      *
    *              DATA BASE CIRECTORY ENTRY - BLDL FORMAT                 *
    *                                                                      *
    ************************************************************************
000 DBDIRB   DSECT
000 DBCIBBGN DS    0H
000 DBDIBFF  DS    AL2                 NUMBER OF BLDL LIST ENTRIES
002 DBDIBLL  DS    AL2                 LENGTH OF EACH BLDL LIST ENTRY
004          ORG   DBCIBBGN
000 DBDIBNAM DS    CL8                 ARRAY NAME
008 DBDIBICB DS    AL3                 ITEM CONTROL BLCCK RECORD TTR
008 DBDIBK   DS    AL1                 CCNCATENATICN NUMBER
00C DBDIBZ   DS    AL1                 LIBRARY TYPE
00D DBDIBC   DS    AL1                 CCUNT - SIZE OF USER DIRECTORY DATA
00E DBDIBDTA DS    AL4                 TTRN - FIRST DATA RECORD
012 DBDIBNUM DS    AL2                 NUMBER OF ITEMS
014 DBDIBBAS DS    AL2                 BLOCK/ARRAY SIZE
016 DBDIBUSE DS    AL1                 USE CODE
    *DBBRES   BIT  7         CN - DA RESIDENT --- OFF - VS RESIDENT
017 DBBRES   DS    XL(B'00000001')
018          ORG   *-B'00000001'
    *DBBBLOCK BIT  6         ON - BLOCKED     --- OFF - UNBLOCKED
017 DBBBLOCK DS    XL(B'00000010')
019          ORG   *-B'00000010'
    *DBBALIGN BIT  5         ON - VS PAGE BOUNCARY ORIGIN
    *                        OFF - VS DOUBLEWCRD BOUNCARY ORIGIN
017 DBBALIGN DS    XL(B'00000100')
01B          ORG   *-B'00000100'
    *DBBMIN   BIT  4         ON - ALIGN TO OCCUPY MINIMUM NUMBER OF VS
    *                             PAGES - VALID ONLY IF BIT 5 IS OFF
    *                        OFF - USE BIT 5 TO DETERMINE BCUNDARY
017 DBBMIN   DS    XL(B'00001000')
01F          ORG   *-B'00001000'
    *DBBINIT  BIT  3         ON - INITIALIZE ARRAY CATA TO VS - VALID
    *                             ONLY IF BIT 7 IS OFF
    *                        OFF - DO NOT INITIALIZE ARRAY CATA TO VS
017 DBBINIT  DS    XL(B'00010000')
027          ORG   *-B'00010000'
    *CBBREINT BIT  2         ON - AFTER RESTART, REINITIALIZE VS ARRAY
    *                             FROM LOG CATA SET - VALID ONLY IF
    *                             BIT 7 IS OFF
    *                        OFF - DO NOT REINITIALIZE ARRAY
017 DBBREINT DS    XL(B'00100000')
037          ORG   *-B'00100000'
    *DBBLOG   BIT  1         ON - LOGABLE VS ARRAY - VALID CNLY IF
    *                             BIT 7 IS OFF
    *                        OFF - ARRAY IS NCT LOGABLE
017 DBBLOG   DS    XL(B'01000000')
057          ORG   *-B'0100000C'
    *DBBDUMMY BIT  0         CN - DUMMY ARRAY - NO INITIALIZATION OR
    *                             PROCESSING WILL BE PERFORMED
    *                        OFF - REAL ARRAY
017 DBBDUMMY DS    XL(B'10000000')
097          ORG   *-B'10000000'
017 DBDIBFLG DC    X'00'
018 DBDIBBCT DS    AL2                 BLOCK COUNT
```

Figure C-30 (2 Of 2). DBDIRB

```
01A DBDIBAID DS    AL2                ARRAY ID
01C DBDIBUPD DS    AL2                UPDATE LEVEL
01E DBDIBEND DS    0H
    DBDIBSIZ EQU   DBDIBEND-DBDIBBGN  SIZE OF DIRECTORY
```

Figure C-31. DBDIRR

```
    ***********************************************************************
    *                                                                     *
    *              DATA BASE DIRECTORY ENTRY - READ FORMAT                *
    *                                                                     *
    ***********************************************************************
000 DBDIRR   DSECT
000 DBDIRBGN DS    0H
000 DBDIRNAM DS    CL8                 ARRAY NAME
008 DBDIRICB DS    AL3                 ITEM CONTROL BLOCK RECORD TTR
00B DBDIRC   DS    AL1                 COUNT - SIZE OF USER DIRECTORY INFO
00C DBDIRDTA DS    AL4                 TTRN - FIRST DATA RECORD
010 DBDIRNUM DS    AL2                 NUMBER OF ITEMS
012 DBDIRBAS DS    AL2                 BLOCK/ARRAY SIZE
014 DBDIRUSE DS    AL1                 USE CODE
    *DBRRES   BIT  7        ON - DA RESIDENT --- OFF - VS RESIDENT
015 DBRRES   DS    XL(B'00000001')
016          ORG   *-B'00000001'
    *DBRBLOCK BIT  6        ON - BLOCKED     --- OFF - UNBLOCKED
015 DBRBLOCK DS    XL(B'00000010')
017          ORG   *-B'00000010'
    *DBRALIGN BIT  5        ON - VS PAGE BOUNDARY ORIGIN
    *                       OFF - VS DOUBLEWORD BOUNDARY ORIGIN
015 DBRALIGN DS    XL(B'00000100')
019          ORG   *-B'00000100'
    *DBRMIN   BIT  4        ON - ALIGN TO OCCUPY MINIMUM NUMBER OF VS
    *                            PAGES - VALID ONLY IF BIT 5 IS OFF
    *                       OFF - USE BIT 5 TO DETERMINE BOUNDARY
015 DBRMIN   DS    XL(B'00001000')
01D          ORG   *-B'00001000'
    *DBRINIT  BIT  3        ON - INITIALIZE ARRAY DATA TO VS - VALID
    *                            ONLY IF BIT 7 IS OFF
    *                       OFF - DO NOT INITIALIZE ARRAY DATA TO VS
015 DBRINIT  DS    XL(B'00010000')
025          ORG   *-B'00010000'
    *DBRREINT BIT  2        ON - AFTER RESTART, REINITIALIZE VS ARRAY
    *                            FROM LOG DATA SET - VALID ONLY IF
    *                            BIT 7 IS OFF
    *                       OFF - DO NOT REINITIALIZE ARRAY
015 DBRREINT DS    XL(B'00100000')
035          ORG   *-B'00100000'
    *DBRLOG   BIT  1        ON - LOGABLE VS ARRAY - VALID ONLY IF
    *                            BIT 7 IS OFF
    *                       OFF - ARRAY IS NOT LOGABLE
015 DBRLOG   DS    XL(B'01000000')
055          ORG   *-B'01000000'
    *DBRDUMMY BIT  0        ON - DUMMY ARRAY - NO INITIALIZATION OR
    *                            PROCESSING WILL BE PERFORMED
    *                       OFF - REAL ARRAY
015 DBRDUMMY DS    XL(B'10000000')
095          ORG   *-B'10000000'
015 DBDIRFLG DC    X'00'
016 DBDIRBCT DS    AL2                 BLOCK COUNT
018 DBDIRAID DS    AL2                 ARRAY ID
01A DBDIRUPD DS    AL2                 UPDATE LEVEL
01C DBDIREND DS    0H
    DBDIRSIZ EQU   DBDIREND-DBDIRBGN   SIZE OF DIRECTORY ENTRY
```

Figure C-32. DBDMPNDR

```
    ***********************************************************************
    *                                                                     *
    *                DUMPLOG HEADER RECORD                                *
    *                                                                     *
    ***********************************************************************
000 DBDMPHDR DSECT
000 DBDMPBGN DS    0F
000 DBDMPAL1 DS    3F                   PRIMARY ALT -- DBALTPRI
00C DBDMPAL2 DS    4F                   SECONDARY ALT -- DBALT SEC
01C DBDMPASZ DS    F                    ARRAY SIZE
020 DBDMPLCB DS    12F                  LOGGING CONTROL BLOCK -- DBLOGCB
050 DBDMPSTM DS    F                    START TIME - FROM DUMPLOG REQUEST
054 DBDMPSDA DS    H                    START DAY -- FROM DUMPLOG REQUEST
056 DBDMPUN1 DS    H                    UNUSED
058 DBDMPETM DS    F                    END TIME - FROM DUMPLOG REQUEST
05C DBDMPEDA DS    H                    END DAY -- FROM DUMPLOG REQUEST
05E DBDMPUN2 DS    H                    UNUSED
060 DBDMPUN3 DS    12F                  UNUSED
090 DBDMPLSR DS    XL256                USER DATA AREA
190 DBDMPEND DS    0F
    DBDMPSIZ EQU   DBDMPEND-DBDMPBGN    SIZE OF DUMPLOG HEADER RECORD
```

Figure C-33. DBITEMD

```
    ***********************************************************************
    *                                                                     *
    *                ITEM   MACRO EXPANSION DSECT                         *
    *                                                                     *
    ***********************************************************************
000 DBITEMD  DSECT
000 DBITSTRT DS    0F
000 DBITNAME DS    CL8                  ITEM NAME
008 DBITLEN  DS    X                    ITEM LENGTH
009 DBITTYPE DS    X                    ITEM TYPE
00A DBITDISP DS    H                    ITEM DISPLACEMENT IN ARRAY
00C DBITAID  DS    H                    ITEM ARRAY ID
00E DBITRPT  DS    H                    ITEM REPITITIONS
010 DBITSTOP DS    0F
    DBITSIZE EQU   DBITSTOP-DBITSTRT
```

Figure C-34. DBLOGCB

```
    ***********************************************************************
    *                                                                     *
    *                LCGGING CCNTRCL ELOCK                                *
    *                                                                     *
    ***********************************************************************
    *                                                                     *
    *  FLNCTICN - CCNTAINS NECESSARY INFCRMATICN TC CCNTFCL LOGGING CF    *
    *             CATA BASE ARRAYS                                        *
    *                                                                     *
    *  REFERENCED BY -  CCNTROL ELCCK = CBALTPRI - LABEL = CBALTLCB       *
    *                                                                     *
    ***********************************************************************
000 CBLOGCB  DSECT
    ***********************************************************************
    *                                                                     *
    *  THESF LAEELS DESCRIBE TFE CATA IN TFE FIRST ENTRY (ENTRY 0) CF     *
    *  THE TABLE.  ENTRY 0 CCNTAINS CONTROL INFORMATICN AND POINTERS.     *
    *                                                                     *
    ***********************************************************************
000 DBLGBGN  DS    0F
000 CBLGNUM  DS    F                          NUMBER CF ENTRIES
004 DBLGFRQ0 DS    A                          LCG FREQ. 0 LIST
008 CBLGFRQ1 DS    A                          LCG FREC. 1 LIST
00C DBLGFRQ2 DS    A                          LOG FREC 2 LIST
010 DBLGFRQ3 DS    A                          LOG FREC.3 LIST
014 DBLGUNLS DS    7F                         UNLSEC
030 DBLGEND  DS    0F
    DBLGESZ  EQU   DBLGEND-CBLGBGN            ENTRY SIZE
030          ORG   CBLGBGN
    ***********************************************************************
    *                                                                     *
    *  TFESE LAEELS DESCRIBE TFE CATA IN ALL SUBSEQUENT ENTRIES CF        *
    *  THE TABLE.  TFERE IS CNE ENTRY FOR EACH LCGABLE ARRAY IN THE       *
    *  DATA BASE.                                                         *
    *                                                                     *
    ***********************************************************************
000 DBLGANAM DS    CL8                        ARRAY NAME
008 DBLGLNAM DS    CL8                        LOG ARRAY NAME
010 DBLGCTIM DS    F                          CURRENT ENTRY TIME
014 DBLGCDAY DS    H                          CURRENT ENTRY CAY
016 DBLGCBLK DS    H                          CURRENT ENTRY BLOCK NUMBER
018 DBLGFTIM DS    F                          FIRST ENTRY TIME
01C DBLGFCAY DS    H                          FIRST ENTRY CAY
01E CBLGAID  DS    H                          ARRAYID
020 DBLGLTIM DS    F                          LAST ENTRY TIME
024 CBLGLCAY DS    H                          LAST ENTRY CAY
026 DBLGLAID DS    H                          LOG ARRAY IC
028 DBLGWNAM DS    CL8                        WRAP ARCUND PROCESSOR NAME
```

Figure C-35. DBLOGHDR

```
    ***********************************************************************
    *                                                                     *
    *               LOGGING HEADER                                        *
    *                                                                     *
    ***********************************************************************
    *                                                                     *
    *  FUNCTION - THE LOG HEADER IMMEDIATELY PRECEEDS THE DATA FOR        *
    *             ALL LOGABLE DATA BASE ARRAYS.  IT CONTAINS DATA         *
    *             NECESSARY FOR PROPER LOGGING AND RETRIEVAL OF ARRAY     *
    *             DATA.                                                   *
    *                                                                     *
    ***********************************************************************
000 DBLOGHDR DSECT
    ***********************************************************************
    *                                                                     *
    *  THESE LABELS DESCRIBE THE LOG HEADER AS IT APPERARS IN VIRTUAL     *
    *  STORAGE AND ON EACH LOGGED COPY OF AN ARRAY.                       *
    *                                                                     *
    ***********************************************************************
000 DBLHBGN  DS    0F
000 DBLHCTIM DS    F                   CURRENT ENTRY TIME
004 DBLHCDAY DS    H                   CURRENT ENTRY DAY
006 DBLHCBLK DS    H                   CURRENT ENTRY BLOCK
008 DBLHBKCT DS    H                   BLOCK COUNT - NUMBER OF BLOCKS
00A DBLHBKEN DS    XL1                 NUMBER BLOCKS PER ENTRY
00B DBLHUPD  DS    XL1                 UPDATE LEVEL
00C DBLHBKSZ DS    H                   BLOCK SIZE
00E DBLHLAID DS    H                   LOG ARRAY ID
010 DBLHPTIM DS    F                   PREVIOUS ENTRY TIME
014 DBLHPDAY DS    H                   PREVIOUS ENTRY DAY
016 DBLHPBLK DS    H                   PREVIOUS ENTRY BLOCK
018 DBLHEND  DS    0F
    DBLHSIZE EQU   DBLHEND-DBLHBGN     LOG HEADER SIZE
    ***********************************************************************
    *                                                                     *
    *  THESE LABELS DESCRIBE THE LOG HEADER AS IT APPEARS ON THE          *
    *  INITIAL DATA FOR EACH LOGABLE DATA BASE ARRAY.                     *
    *                                                                     *
    ***********************************************************************
018          ORG   DBLHBGN
000 DBLHANAM DS    CL8                 LOG ARRAY NAME
008 DBLHWNAM DS    CL8                 LOG WRAP ARROUND PROCESSOR
010 DBLHCOPY DS    H                   LOG COPIES
012 DBLHFREQ DS    XL1                 LOG FREQUENCY
013 DBLHBKCP DS    XL1                 BLOCKS PER COPY
```

Figure C-36. DBPBT

```
    ************************************************************************
    *                                                                      *
    *              PAGE BOUNDARY TABLE                                     *
    *                                                                      *
    ************************************************************************
    *                                                                      *
    *  FUNCTION - CONTAINS ADDRESSES OF ENTRIES WITHIN DBALTSEC THAT       *
    *             ARE ON VIRTUAL STORAGE PAGE BOUNDARIES.                  *
    *                                                                      *
    *  REFERENCED BY -  CONTROL BLOCK = SCVT     - LABEL = SCVTAPBT        *
    *                   CONTROL BLOCK = DBALTSEC - LABEL = DBALTPBT        *
    *                                                                      *
    ************************************************************************
000 DBPBT    DSECT
    ************************************************************************
    *                                                                      *
    *  THESE LABELS DESCRIBE THE DATA CONTAINED IN EACH ENTRY OF THE       *
    *  TABLE.  THE FIRST ENTRY POINTS TO THE FIRST ARRAY ENTRY IN THE      *
    *  DBALTSEC CONTROL TABLE.  EACH SUBSEQUENT ENTRY POINTS TO THE        *
    *  LAST DBALTSEC ENTRY ON EACH VIRTUAL STORAGE PAGE OCCUPIED BY THE   *
    *  DBALTSEC.  THE LABEL DBALTPBS IN THE DBALTSEC CONTAINS THE          *
    *  SIZE OF THE PAGE BOUNDARY TABLE.                                    *
    *                                                                      *
    ************************************************************************
000 DBPBTBGN DS    0F
000 DBPBTNAM DS    CL8                  ARRAY NAME
008 DBPBTALT DS    F                    A(SECONDARY ALT ENTRY)
00C DBPBTEND DS    0F
    DBPBTESZ EQU   DBPBTEND-DBPBTBGN    PBT ENTRY SIZE
```

Figure C-37 (1 Of 3). WAREA

```
    **********************************************************************
    *                                                                    *
    *                                                                    *
    *             DATA BASE FINAL PHASE PRCCESSCR WCRK AREA              *
    *                                                                    *
    *                                                                    *
    **********************************************************************
000 WAREA     DSECT
000 AREASTRT  DS    0D
000 REGSAVE   DS    9D
048 WAINPARM  DS    F                      A(INPUT PARM LIST)
04C WARTNCOD  DS    H                      FPP RETURN CODE
C50 WASTART   DS    0D
050 WABLKPCS  DS    F                      PCS BLKSIZE
054 WABLKCDS  CS    F                      DCS BLKSIZE
058 WABLKNUM  DS    F                      HIGEST BLCCK NUMBER USEC
05C WANXTBLK  CS    F                      A(WHERE NEXT OUTPUT BLK TC BE PUT)
060 WAPRTMVC  DS    F                      SIZE OF PARTIAL BLCCK MOVED
064 WABLKPTR  CS    F                      A(CURRENT BLOCK CCNTROL BLCCK)
068 WAITMCNT  DS    F                      NUMBER CF ITEMS IN CBDEF
06C WANEWCID  DS    F                      A(NEW CID ITEM CCNTRCL BLCCKS
070 WADELTAB  DS    F                      A(DELETE AID TABLE)
074 WADLTBSZ  DS    F                      SAVE SIZE OF DELETE TABLE
C78 WADELARR  DS    F                      NUMBER OF DELETED ARRAYS
C7C WANXTDEL  DS    F                      NEXT CELETE TABLE ENTRY
C80 WANEWAID  CS    F                      NEW AIC CCUNTER
084 WANXTDIR  DS    F                           NEXT AVAIAABLE NEW DIR ENT
C88 WANEWCIR  DS    F                           NEW DIRECTORY ENTRY TABLE
C8C WANDIRSZ  DS    F                           SIZE OF NEW DIR ENT TABLE
090 WACICBLK  DS    H                      CICS BLCCK COUNT
092 WAREFBLK  DS    H                      REFRESH ARRAY BLOCK COUNT
C94 WAICBBUF  DS    F                      A(2048 BYTE OUTPUT BUFFER)
098 WAOLCCID  DS    F                      A(2048 BYTE INPUT  BUFFER)
09C WAICBTTR  DS    F                      SAVE ICB RECORD TTR
0A0 WADTATTR  DS    F                      SAVE CATA RECCRD TTR
0A4 WAREITTR  DS    F                      REFRESH ARRAY ICB TTR
0A8 WAREDTTR  DS    F                      REFRESH ARRAY DATA TTR
0AC WANEGAID  DS    H                      NEGATIVE ARRAY ID
0B0           CS    0F                     *
0B0           DS    H                      *  ALIGN TO HALFWORC BOUNCARY
0B2 WADBCIR   DS    32X                    *  BLDL/STOW AREA
0D2 WAREFDIR  CS    32X                    REFRESH ARRAY DIRECTORY ENTRY
0F4 WASAVR2   DS    F                      SAVE REG 2
0F8 WASAVR3   DS    F                      SAVE BAL REG 3
0FC WASAVR4   DS    F                      SAVE BAL REG 4
100 WASAVR5   DS    F                      SAVE REG 5
104 WASAVR9   DS    F                      SAVE A(FIRST ARRAY)
108 WANUMARR  DS    F                      OLD ARR + NEW ARR = TOTAL DB ARRAYS
10C WANUMNEW  DS    F                      NUMBER OF NEW ARRAYS
110 WANUMDEL  DS    F                      NUMBER OF DELETEC ARRAYS
114 WAUNSBLK  DS    F                      NUMBER CF UNUSED DIRECTORY BLCCKS
118 WANUMBLK  DS    F                      NUMBER OF DIRECTORY BLOCKS
11C WADIRIN   DS    F                      FIRST INPUT DIRECTORY BLOCK
120 WADIROUT  DS    F                      FIRST OUTPUT DIRECTCRY BLOCK
124 WACIRISZ  CS    F                      DIRECTORY INPUT SIZE
128 WADIROSZ  DS    F                      DIRECTORY OUTPUT SIZE
12C WADECBSZ  DS    F                      SIZE CF DECB STORAGE
```

Figure C-37 (2 Of 3). WAREA

```
130 WACECBAD DS     F                             A(DECE STORAGE)
134 WAAIDSIZ DS     F                             SIZE OF GETMAIN FOR AID TABLE
138 WAAICTAB DS     F                             A(AID UPDATE TABLE)
13C WAZFROID DS     F                             A(ZERO AID IN AID TABLE)
140 WAINTTTR DS     F
144 WAINTBUF CS     8F
164 WALMCBDA DS     F                             A(DPPXCBCA)
168 WALMCBLG CS     F                             A(DPPXDBLG)
16C WADABUF  DS     4F                            DA CNTL RECORD BUFFER
17C WANUMCLG DS     F                             NUMBER OF OLD LOGABLE ARRAYS
180 WANUMNLG CS     F                             NUMBER OF NEW LOGABLE ARRAYS
184 WANEWREF DS     F                             A(NEW REFRESH TABLE)
188 WANREFSZ DS     F                             SIZE CF NEW REFRESH TABLE
18C WANXTREF DS     F                             A(NEXT REFRESH TABLE ENTRY)
190 WAFSTCTA DS     XL(B'1CC000CC0')
210          ORG    *-B'1CC000CC'
190 WADELETE DS     XL(B'00100000')
180          ORG    *-B'0010C00C'
190 WATFST   CS     XL(B'00010000')
1A0          CRG    *-B'00010000'
190 WAARCLET DS     XL(B'0C001000')
198          ORG    *-B'00001000'
190 WAFSTICB DS     XL(B'C0000100')
194          ORG    *-B'00000100'
190 WANEWPDS DS     XL(B'00000010')
192          ORG    *-B'C0000010'
190 WADIRECF DS     XL(B'C0000001')
191          ORG    *-B'00000001'
190 WAFLG1   DC     X'00'
191 WAREPL   DS     XL(B'10000000')
211          ORG    *-B'1CC00000'
191 WADIREND DS     XL(B'01C00000')
1D1          ORG    *-B'01000000'
191 WAPRTBLK CS     XL(B'00100000')
1B1          ORG    *-B'0010000C'
191 WAINCCIR DS     XL(B'00010000')
1A1          ORG    *-B'00010000'
191 WAFSTCIC DS     XL(B'C0001000')
199          CRG    *-B'00001000'
191 WANFSCID DS     XL(B'00C00100')
195          ORG    *-B'0000010C'
191 WAWRCICS DS     XL(B'00000010')
193          ORG    *-B'0000001C'
191 WAENDCID DS     XL(B'C0000001')
192          ORG    *-B'00C00001'
191 WAFLG2   DC     X'00'
192 WATSTERR DS     XL(B'10000000')
212          ORG    *-B'1C000000'
192 WAPRINT  DS     XL(B'C1000000')
1D2          ORG    *-B'01000000'
192 WASNTKRD DS     XL(B'001000C0')
182          ORG    *-B'0010000C'
192 WARC04   DS     XL(B'00010000')
1A2          ORG    *-B'00010000'
192 WAFLAG3  DC     X'00'
198 WAPACK   DS     D
1A0 WAPDSCCB DS     F                             A(CBASPCS DCB)
```

Figure C-37 (3 Of 3). WAREA

```
1A4 WARCDCB   DS    F                      A(CBASPCSR DCB)
1A8 WADECBRD  DS    F                      A(CECBINTU)
1AC WAMESAGE  DS    F                      A(MESSAGE TO PRINT)
1B0 WADEVTYP  DS    6F                     DEVTYP RETURN AREA
1C8 WACACNTR  DS    F                      CA ARRAY COUNTER          DR 5203
1D0 WASTOP    DS    0D
    WASIZE    EQU   WASTOP-WASTART
1D0 WALINE    DS    CL121                  HOLDS 1 LINE FOR SYSPRINT
250 AREASTOP  DS    0D
    AREASIZE  EQU   AREASTOP-AREASTRT
```

Figure C-38. BRT1

```
    *
    *      GENERATE CSECTS FOR BRANCH TABLES USED TO OBTAIN                 *
    *      ENTRY POINT OF SRTOS SVC ROUTINES FOR BRANCH ENTRY               *
    *
000 BRT1      CSECT
000           DC    A(0)                   NC-NC       UNUSED
004 BRT1PSVC  CC    A(0) .  ID=04 X'04'    PATCH       EP=DPPTPSVC
008 BRT1CSVC  DC    A(0) .  ID=C8 X'08'    CPATCH      EP=DPPTDSVC
00C BRT1CSVC  DC    A(0) .  ID=12 X'0C'    CHAIN       EP=DPPTCSVC
010 BRT1WSVC  CC    A(0) .  ID=16 X'10'    GET/FREWA   EP=DPPTWSVC
014 BRT1SVCP  DC    A(0) .  ID=20 X'14'    SETPSW      EP=DPPXSVCP
018 BRT1CBCT  CC    A(0) .  ID=24 X'18'    CB-GET/FRE  EP=DPPTCBGT
```

Figure C-39. BRT2

```
    *
    *
    *
000 BRT2      DSECT
000           DC    A(0) .                 NO-NO       UNUSED
004 BRT2PTIM  DC    A(0) .  ID=04 X'04'    PTIME       EP=DPPCTSVC
008 BRT2PGFX  CC    A(0) .  ID=C8 X'C8'    PAGFIX      EP=DPPFIXFR
00C BRT2PGFR  DC    A(0) .  ID=12 X'0C'    PAGFREE     EP=DPPFIXFR
010 BRT2RSVC  DC    A(0) .  ID=16 X'10'    REPATCH     EP=DPPTRSVC
014 BPT2WQDL  DC    A(0) .  ID=20 X'14'    WQE-DELETE  EP=DPPTWQDL
```

Figure C-40. DRT

```
000 DRT       DSECT   DATA RECORDING TABLE
    **********************************************************************
    *  THE DATA RECORDING TABLE(DRT) IS AN INCORE TABLE WHICH CONTAINS A *
    *  COPY OF THE SRTOS TIME ARRAY,VALID DATA RECORDING ID'S AND THE    *
    *  OPENED DATA RECORDING DCB.THE TABLE IS BUILT BY THE DATA RECORDING*
    *  INITIALIZATION ROUTINE. THE TABLE IS POINTED TO BY THE SCVT AT    *
    *  LOCATION SCVTRWA                                                  *
    **********************************************************************
000 DRTIME    DS    0CL28                  THE TIME ARRAY IS 28 BYTES
000 DRTHS     DC    F'0'                   TOD 10 MILLISEC
004 DRTTOD    DC    F'0'                   TOD IN DECIMAL
008 DRTJDAY   DC    F'0'                   JULIAN DATE
00C DRTMDAY   DC    F'0'                   DAY OF MONTH
010 DRTEBC    DC    CL10' '                DATE IN EBCDIC
01A DRTBDAY   DC    H'0'                   BINARY DAY OF YEAR
01C DRTCT     DC    H'0'                   COUNT OF ENABLED ID'S OR X'FF' IF
    *                                      ALL WAS SPECIFIED ON ENABLE COMMAND
01E DRTID     DC    50H'0'                 UP TO 50 ENABLED ID'S-EACH 2 BYTES
    DRTRLEN   EQU   *-DRT                  THIS PART IS PUT OUT ON TIME HEADER
084           DS    0F                     FORCE WORD BOUNDRY
084 DRTDCB    DC    XL100'0'               SPACE FOR A DCB
0E8 DRTDREC   DC    A(0)                   ADDRESS OF DATA RECORDING MODULE
0EC DRTEXIT   DS    0F                     DCB EXIT LIST
0EC DRTEXABF  DS    0X                     FLAG BYTE FOR ABEND EXIT ADDR
0EC DRTEXABA  DC    F'0'                   ABEND EXIT ADDR
0F0 DRTEXDCF  DS    0X                     FLAG BYTE FOR DCB EXIT
0F0 DRTEXDCB  DC    F'0'                   DCB EXIT ADDRESS
0F4 DRTDRCX   DC    F'0'                   ADDRESS OF DUMMY DATA RECORD MODULE
0F8 DRTEND    DS    0F
    DRTLEN    EQU   DRTEND-DRT             LENGTH OF AREA
```

Figure C-41. GFCB

```
000 GFCB      DSECT
    *
    **   GFCB  =  GETWA/FREEWA CONTROL BLOCK
    *             - INITIALLY - ONE FOR EACH GETWA SIZE
    *             - CHAIN MAY BE EXPANDED BY CPPTGWFW
    *             - GFCB'S ARE MAINTAINED ON 2 CHAINS - GFMB (GFMBGFCB)
    *                                                 - TMCT (TMCTGFCB)
    *
000 GFCBFLGS DC     X'0'       FLAG BYTE
001          ORG    GFCBFLGS
000 GFCBGFMB DC     A(0) BKWD POINTER TO GFMB
004 GFCBNEXT DC     A(0) POINTER TO NEXT GFCB ON CHAIN ORIGINATING TMCTGFCB
008 GFCBFRST DC     A(0) ADDRESS OF FIRST CORE LOC ALLOCATED FOR THIS SIZE
00C GFCBLAST DC     A(0) ADDRESS OF  LAST CORE LOC ALLOCATED FOR THIS SIZE
010 GFCBGFBE DC     A(0) POINTER TO FIRST FREE GFBE IN FREE CHAIN
014 GFCBGFCB DC     A(0) POINTER TO NEXT GFCB ON CHAIN ORIGINATING GFMB
018 GFCBFCNT DC     H'0' # FREE BLOCKS CONTROLLED BY THIS GFCB
01A GFCB#BLK DC     H'0' # BLOCKS CONTROLLED BY THIS GFCB
    *
    *  GFCBFLGS  DEFINITIONS
    GFCBINIT EQU    X'80'      INITIALLY ALLOCATE GETWA SPACE
    GFCBFREE EQU    X'40'      ALL BLOCKS FOR THIS GFCB FREE
    *
    *
    GFCBLNTH EQU    *-GFCB     LENGTH OF GFCB
    *
    *
```

Figure C-42. GFBE

```
000 GFBE      DSECT
    *
    **   GFBE  =  GETWA/FREEWA BLOCK ENTRY - ONE FOR EACH BLOCK OF CORE
    **                                       REPRESENTED BY GFCB - APPENDED
    **                                       ON END OF GFCB
    *             - WHEN BLOCK IS FREE GFBE IS ON FREE CHAIN (GFCBFREE)
    *             - WHEN BLOCK IS ALLOCATED - GFBE WILL BE ONE OF THE
    *             -       FOLLOWING CHAINS - TYPE = PC - TMCTEXGW
    *                                      - TYPE = AT - TCBXTGWA
    *                                      - TYPE = AP - TCBXQGWA
    *
000 GFBEFLAG DC     X'00'      FLAGS : 01 = BLOCK ALLOCATED
001          ORG    GFBEFLAG
000 GFBENEXT DC     A(0)       ALLOCATED= PTR TO NEXT GFBE IN CIRCULAR CHAIN
    *                          NOT ALLOC= PTR TO NEX FREE GFBE
004 GFBEID   DC     X'00'      ALLOCATED= ID OF ASSOCIATED GFMB
005          ORG    GFBEID
004 GFBEPREV DC     A(0)       ALLOCATED= PTR TO PREV GFBE IN CIRCULAR CHAIN
    *                          NOT ALLOC= MEANINGLESS
    GFBELNTH EQU    *-GFBE     LENGTH OF GFBE
```

Figure C-43. IMP

```
000 IMP         DSECT   INPUT MESSAGE PROCESSING TABLE
    *  THE INPUT MESSAGE PROCESSING TABLE(IMP) IS AN INCORE TABLE WHICH  *
    *  CONTAINS THE VALID SRTOS SYSTEM OPERATOR CODES.THE IMP CONTAINS   *
    *  INFORMATION UTILIZED BY THE SRTOS INPUT MESSAGE ROUTINES.IT IS    *
    *  INCLUDED AS ARRAY DPPXIMP IN THE DATA BASE AT SYSTEM BUILD TIME    *
000 IMPCNT   DC     XL2'0'                NUMBER OF ENTRIES IN IMP
    * THE FOLLOWING PARAMETERS CONSTITUTE AN ENTRY *
002 IMPCODE  DC     CL8' '                THE ENTRY CODE
00A IMPTASK  DC     CL8' '                TASKNAME OF TASK TO BE PATCHED
012 IMPLM    DC     CL8' '                ENTRY POINT NAME OF THE LM TO PATCH
01A IMPID    DC     XL1'0'                ID ASSIGNED LOAD MODULE WHEN PATCHED
01B IMPLGTH  DC     XL1'0'                NUMBER OF BYTES IN ENTRY
    * THE NUMBER OF PARAMETER CONVERSION CODES & LENGTHS IS EQUAL TO THE
    * NUMBER OF PARAMETERS THAT MAY BE PASSED TO THE SPECIFIED LOAD MODULE
01C IMPCCN   DC     CL1' '                CONVERSION CODES OF THE PARAMETERS
    *                                       THAT MAY BE PASSED TO THE LOAD
    *                                       MODULE. C-CHARACTER DATA
    *                                       X-HEXADECIMAL DATA F-FULLWORD DATA
01D IMPARMLN DC     XL1'0'                LENGTH OF THE PASSED PARAMETER
```

Figure C-44. LCB

```
000 LCB       DSECT
    *
    ** LCB = LOAD CONTROL BLOCK - CHAINED TO TCBX
    *
000 LCBNEXT  DC     A(0) .    POINTER TO NEXT LCB IN CHAIN
004 LCBLCBA  DC     A(0)      ADDRESS OF ASSOCIATED LCB ON TMCT-LCB RENT CH
008 LCBFLAGS DC     X'0' .    FLAG BITS
009 LCBATR1  DC     X'0' .    ATTRIBUTES MOVED IN FROM PDS2ATR1 AFTER BLDL
00A LCBUSFCT DC     H'0'      USE COUNT FOR LCB ON TMCT-LCB RENT CHAIN
00C          ORG    LCBUSECT
00A LCBREQCT DC     H'0'      REQUEST COUNT FOR LCB ON TCBX-LCB CHAIN
00C LCBEPAD  DC     A(0)      ENTRY POINT ADDRESS
010 LCBEPNAM DC     CL8' ' .  ENTRY POINT NAME
018 LCBBLDL  DC     A(0) .    ADDRESS OF BLDL TABLE FOR LOAD BY DPPTSMON
01C LCBECBAD DC     A(0)      ADDRESS OF ECB THAT DPPTDLMP WAITS ON
    LCBLNTH  EQU    *-LCB     LENGTH OF LCB
    *  LCBFLAGS EQU'S
    LCBFUNRS EQU    X'01'     LCB UNRESOLVED BIT
    LCBFTMCT EQU    X'04'     THIS LCB IS CHAINED TO A TMCT-LCB CHAIN
    LCBFLOAD EQU    X'08'     LOAD PROCESSING BY DPPTSMON REQUESTED
    LCBFDEL  EQU    X'10'     DELETE PROCESSING BY DPPTSMON REQUESTED
    LCBFLMP  EQU    X'20'     LOAD MODULE PURGE REQUESTED
    *  LCBATR1 EQU'S    - THESE ATTRIBUTES ARE MOVED IN FROM PDS2ATR1
    LCBARENT EQU    X'80'     REENTERABLE
    LCBAREUS EQU    X'40'     REUSABLE
    LCBAOVLY EQU    X'20'     IN OVERLAY STRUCTURE
    LCBAOL   EQU    X'C8'     ONLY LOADABLE
    LCBAEXEC EQU    X'02'     EXECUTABLE
```

Figure C-45. LOCKCBLK

```
000 LOCKCBLK DSECT
    ***
    *         LOCK CONTROL BLOCK
    ***
000 LOCKNEXT DC    A(0)                  ADDRESS OF NEXT LOCKCBLK
004 LOCKNAME DC    CL4' '                RESOURCE NAME
008 LOCKFLAG DC    X'00'                 RESOURCE RESERVE FLAG
009          ORG   LOCKFLAG
008 LOCKTCBA DC    A(0)                  ADDRESS OF TASK CONTROLING RESOURCE
00C LOCKNFLG DC    X'0'
00D          ORG   LOCKNFLG
00C LOCKWAIT DC    A(0)                  ADDRESS OF WAIT CONTROL BLOCK
010 LOCKXCVT DC    A(0)                  A(XCVT)
014 LOCKCNT  DC    H'0'                  NO.OF DEFLOCK'S FOR THIS RESOURCE
018 LOCKEND  DS    0D
    LOCKLNTH EQU   LOCKEND-LOCKCBLK
```

Figure C-46. WAITCBLK

```
000 WAITCBLK DSECT
000 WAITFLAG DC    X'0'                  RESOURCE RESERVE FLAG
001          ORG   WAITFLAG
000 WAITNEXT DC    A(0)                  NEXT WAIT CONTROL BLOCK
004 WAITECB  DC    F'0'                  WAIT ECB
    WAITEND  EQU   *
    WAITLNTH EQU   WAITEND-WAITCBLK
```

Figure C-47. RCT

```
000 RCT       DSECT    MESSAGE ROUTING CODE TABLES
    ***********************************************************************
    *  THE ROUTING CODE TABLE(RCT) IS AN INCORE TABLE WHICH CONTAINS THE *
    *  CODES OF THE VALID SYSTEM MESSAGES DEVICES.THE ROUTING CODE TABLE *
    *  CONTAINS INFORMATION UTILIZED BY THE SRTOS REAL TIME MESSAGE      *
    *  HANDLER.IT IS INCLUDED AS ARRAY DCMXSMRC IN THE DATA BASE          *
    ***********************************************************************
000 RCTWTO    DC       XL4'0'                   WTO DESCRIPTOR AND ROUTING CODES
004 RCTCNT    DC       XL2'0'                   NUMBER OF ENTRIES IN TABLE
006 RCTDDCNT  DC       XL2'0'                   NUMBER OF DDNAMES IN TABLE
008 RCTEPCNT  DC       XL2'0'                   NUMBER OF ENTRY POINT NAMES IN TABLE
    * THE ENTRY PARAMETERS ARE THE RCTRC RCTGROUP RCTUNIT RCTDEVCE AND THE
    * RCTALTRC PARAMETERS.
00A RCTRC     DC       XL1'0'                   ROUTING CODE ASSIGNED TO A DEVICE
00B RCTGROUP  DC       XL1'0'                   FOR A DISPLAY DEVICE AN ACCESS AREA
    *                                           IS ASSIGNED.FOR AN OS DEVICE AN
    *                                           INDEX INTO THE DDNAME FIELD IS
    *                                           ASSIGNED.FOR A LOAD MODULE AN INDEX
    *                                           INTO THE ENTRY POINT FIELD IS
    *                                           ASSIGNED.
00C RCTUNIT   DC       XL1'0'                   THIS FIELD ONLY HAS MEANING FOR A
    *                                           DISPLAY DEVICE.IT IS SET TO ZERO FOR
    *                                           ALL OTHER DEVICES.THE FUNCTIONAL
    *                                           AREA OF A DISPLAY IS ASSIGNED.
00D RCTDEVCE  DC       XL1'0'                   BITS 0-3 DESIGNATES A
    *                                           FUNCTIONAL OR NON-FUNCTIONAL DEVICE
    *                                           FLAG.1-FUNCTIONAL 0-NON-FUNCTIONAL
    *                                           BITS 16-31 DESIGNATES THE DEVICE
    *                                           1-SYSTEM CONSOLE 2-OS DEVICE
    *                                           3-DISPLAY DEVICE 4-RESERVED
    *                                           5-LOAD MODULE
00E RCTALTRC  DC       XL1'0'                   AN ALTERNATE ROUTE CODE TO WHICH
    *                                           MESSAGE CAN BE PASSED
00F RCTDDNAM  DC       CL8' '                   OS DEVICE DDNAME TO WHICH MESSAGE
    *                                           CAN BE ASSIGNED.
017 RCTEP     DC       CL8' '                   ENTRY POINT OF A LOAD MODULE TO
    *                                           WHICH MESSAGES CAN BE PASSED
```

Figure C-48. MDT

```
000 MDT       DSECT    MESSAGE DCB TABLE
    ***********************************************************************
    *  THE MESSAGE DCB TABLE(MDT) IS AN INCORE TABLE WHICH CONTAINS THE  *
    *  OPENED DCB'S USED BY THE SRTOS MESSAGE HANDLER. MDT IS BUILT BY   *
    *  THE MESSAGE HANDLER INITIALIZATION PROGRAM. THE TABLE IS POINTED   *
    *  TO BY THE MESSAGE ADDRESS TABLE(MCAT).                              *
    ***********************************************************************
000 MDTMSG    DC       CL94' '                  OPENED MESSAGE DCB
    * ANY NUMBER OF QSAM OUTPUT DCB'S CAN BE SPECIFIED *
05E MDTQSAM1  DC       CL96' '                  OPENED QSAM OUTPUT DCB
0BE MDTQSAM2  DC       CL96' '                  OPENED QSAM OUTPUT DCB
11E MDTQSAM3  DC       CL96' '                  OPENED QSAM OUTPUT DCB
17E MDAQSAM4  DC       CL96' '                  OPENED QSAM OUTPUT DCB
```

Figure C-49. MDAT

```
000 MDAT      DSECT    MESSAGE ADDRESS TABLE
    ****************************************************************
    *  THE MESSAGE ADDRESS TABLE(MDAT) IS AN INCORE TABLE WHICH CONTAINS *
    *  THE ADDRESS'S OF LOCK CONTROL BLOCKS,MSG RCT, OPENED DCB'S AND    *
    *  OTHER INFORMATION UTILIZED BY THE SRTOS REAL TIME MESSAGE HANDLER *
    *  IT IS BUILT BY THE MESSAGE HANDLER INITIALIZATION ROUTINE.        *
    *  THE TABLE IS POINTED TO BY THE SCVT AT LOCATION SCVTMWA.           *
    ****************************************************************
000 MDATCNT   DC      XL1'0'                NUMBER OF QSAM(OS DEVICE)DCB
    *                                         ADDRESS'S SPECIFIED
001 MDATFLG   DC      XL1'0'                RESTART FLG
    *                                       0- PRE RESTART 1- POST RESTART
002 MDATRES   DC      XL2'0'                RESERVE BYTE
004 MDATLCK   DC      A(0)                  ADDRESS SRTOS LOCK CONTROL BLOCK
    *                                       UTILIZED BY MESSAGE MACRO PROCESSOR
    *                                       (DPPMMSG)
008 MDATLCKO  DC      A(0)                  ADDRESS SRTOS LOCK CONTROL BLOCK
    *                                       UTILIZED BY MESSAGE OUTPUT ROUTINE
    *                                       (DPPMMSG1)
00C MDATRCT   DC      A(0)                  ADDRESS OF MESSAGE ROUTING CODE
    *                                       TABLE
010 MDATMDCB  DC      A(0)                  ADDRESS OPENED MESSAGE DATA SET DCB
014 MDATQDCB  DC      A(0)                  ADDRESS OF OPENED QSAM OUTPUT DCB
```

Figure C-50. PTIMEL

```
000 PTIMEL    DSECT
    ***                                                               ***
    *         PTIME INPUT PARAMETERS                                    *
    *                                                                   *
    *               REG 1 = ADDR.OF SUPERVISOR LIST (IF REG 0 > ZERO)   *
    *               REG 0 = 0 => RET OPTION                             *
    *                     = 4 => ADD OPTION                             *
    *                     = 8 => MOD OPTION                             *
    *                     = 12=> DEL OPTION                             *
    ***                                                               ***
000 PTIMSFLG DC       XL1'0'                TIME OPTION FLAG
001 PTIMSTRT DC       AL3(0)                START TIME VALUE(OR ADDRESS)
004 PTIMIFLG DC       XL1'0'                PURGE OPTION FLAG
005 PTIMINTL DC       AL3(0)                INTERVAL TIME VALUE(OR ADDRESS)
008 PTIMEFLG DC       XL1'0'                TIME OPTION FLAG
009 PTIMSTOP DC       AL3(0)                STOP TIME VALUE(OR ADDRESS)
00C          ORG      PTIMSTOP
009 PTIMCNT  DC       AL3(0)                COUNT VALUE
00C PTIMPTCH DC       A(0)                  PATCH SUPERVISOR LIST
010 PTIMPARM DC       A(0)                  PATCH PROBLEM PARAMETER LIST
014 PTIMPTQE DC       A(0)                  PTQE ADDR FOR MOD OR DEL
    PTIMLNGH EQU      *-PTIMEL
    *                 PURGE OPTION FLAGS
    PTIMFPRG EQU      X'01'                 PURGE DPATCH = U
    PTIMFDPC EQU      X'02'                 PURGE DPATCH = C
    PTIMFDPW EQU      X'04'                 PURGE DEPATCH = W
    PTIMFADR EQU      X'40'                 PTQE ID INCLUDED
    *                 TIME OPTION FLAGS
    PTIMCFG  EQU      X'08'                 THIS FIELD CONTAINS COUNT VALUE
    PTIMREL  EQU      X'01'                 RELATIVE TIME
    PTIMTOD  EQU      X'02'                 TOD TIME
    PTIMADJ  EQU      X'04'                 ADJUSTED TIME
    PTIMADDR EQU      X'80'                 THIS FIELD CONTAINS TIME ADDRESS
    PTIMLN   EQU      *-PTIMEL
```

Figure C-51. PTQE

```
000 PTQE      DSECT
    *
    ** PTQE = PTIMER QUEUE ELEMENT
    *
    *         A PTQF IS CREATED IN RESPCNSE TC A PTIME MACRO CALL.
    *         THE CHAIN OF PTQE'S IS POINTED TC BY THE SCVT (SCVTTQET).
    *
    *
000 PTQENEXT DC    A(0)        -> NEXT PTQE IN CHAIN
004 PTQFTIME CC    A(0)        -> TIME OF NEXT PATCH (10 MIL UNITS)
008 PTQFINVL DC    A(0)        INTERVAL FOR REPEATING PTQES
00C PTQECNT  DC    H'0'        CCUNT VALUE (# PATCHES UNTIL STOP)
00E PTQEFLG1 CC    X'0'        FLAG BYTE 1
00F PTQEFLG2 DC    X'0'        FLAG BYTE 2
010 PTQEPARM DC    A(0)        -> PRCBLEM PARAMETERS
014 PTQFTASK DC    CL8' '      PATCH TASK NAME
01C PTQEEP   DC    CL8' '      ENTRY PCINT NAME
024 PTQFPREF DC    CL8' '      PRTY REFFRENCE NAME
02C PTQEFLAG DC    X'0'                  PATCH FLAGS
02D PTQECL   DC    X'0'        QUEUE LENGTH
02E PTQFPRTY DC    H'0'                  PRTY VALUE
030 PTQEECB  DC    A(0)        ECB
034 PTQEFREL DC    A(0)        FREE LENGTH
038 PTQEFREA CC    A(0)                  FREE ADDRESS
03C PTQETCBX DC    A(0)                  TCBX
040 PTQEPRBL DC    2F'0'                 PATCH PROBL(IF LESS THAN 8 BYTES)
048 PTQFORCS DC    F'0'                  ORIGINAL START TIME
    *
    **        PTQE  FLAG BYTE 1 FLAGS
    *
    PTQEDEL  ECU   X'01'                 DELETE THIS PTQE
    PTQFFREE EQU   X'02'                 FREE PARM LIST
    PTQEINF  FQU   X'04'                 INFINITE PTIME FLAG
    *
    **        PTQE  FLAG BYTE 2 FLAGS
    *
    PTQEDPU  EQU   X'01'                 CPATCH=U
    PTQFDPC  EQU   X'02'                 DPATCH=C
    PTQFDPW  FQU   X'04'                 DPATCH=W
    PTQELNTH ECU   *-PTCENEXT      LENGTH OF PTCE
```

Figure C-52. PWQE

```
000 PWQE      DSECT
    ***
    *         PWQE -CSECT USED TO DESCRIBE INPLT PARAMETERS TO PURGEWC
    ***
000 PWQETASK  DC     A(0)                    A(TASK NAME) CR ZERO(SELF)
004 PWQEEP    DC     A(0)                    A(ENTRY PCINT NAME)
008 PWQEFCB   DC     A(0)                    A(FCB) TC BE POSTEC - OPT=(POST,ECB)
    *                                        ZERO                - OPT=NOPCST
    *                                        X'8C000000'         -OPT=WAIT
00C PWQELNTH  DC     A(0)                    FREE= LENGTH
010 PWQEACCR  DC     A(0)                    FREE= ACDR
014 PWQEPTN   DC     X'0'                    PARTITICN FLAGS
    *                                        X'00' - PTN=CWN
    *                SUPFPTNS                X'20' - PTN=SLAVE
    *                SUPFPTNM                X'40' * PTN=MASTER
    *                                        X'60' - PTN=FIND
015 PWQEUNUS  DC     X'0'                    UNUSED
016 PWQEID    DC     H'0'                    WQE ID
```

Figure C-53. REPL/SUPL

```
    *
    ** PATCH INPUT PARAMETERS :
    *     REG 1  ADDR OF SUPERVISOR PARAMETER LIST  -SUPL
    *     REG 0  ADDR OF PROBLEM PARAMETER LIST  -PROBL
    *
    ** REPATCH INPUT PARAMETERS :
    *     REG 1  ADDR OF REPATCH PARAMETER LIST  -REPL
    *     REG 0  TYPE INDICATION
    *
    ** SUPERVISOR PARAMETER LIST / REPATCH PARAMETER LIST FORMAT :
    *
000          DSECT
000 REPL     DS     0F
000 SUPL     DS     0F
000 SUPTASK  DC     CL8' '     TASK NAME
008 SUPEP    DC     CL8' '     ENTRY POINT NAME
010 SUPPRTYN DC     CL8' '     PATCH ONLY - PRTY REFERENCE NAME
018 SUPFLAG  DC     X'00'      FLAG BYTE - SEE EQU'S BELOW
019 SUPQL    DC     X'00'      QUEUE LENGTH
01A SUPPRTYV DC     H'0'       PRTY RELATIVE VALUE
01C SUPECB   DC     A(0)       ECB ADDRESS
020 SUPFREEL DC     A(0)       FREE LENGTH
024 SUPFREEA DC     A(0)       FREE ADDRESS
028 SUPTCBX  DC     A(0)       TCBX
    SUPLLNTH EQU    *-SUPL
    *
    *  EXTENSION TO SUPL  TO FORM REPATCH PARAMETER LIST :
    *
02C REPLPARM DC     A(0)       ADDR OF PROBL ASSOC WITH THIS REPL
030 REPLPROB DC     XL8'0'     PROBL MOVED HERE IF LE 8 BYTES LONG
038 REPLAD   DC     A(0)       ADDR OF REPL
    REPLLNTH EQU    *-REPL     LENGTH OF REPL TO BE USED BY PROBLEM PROGRAMS
03C REPLCHN  DC     A(0)       CHAIN WORD FOR SUPPLIED REPL'S ONLY
040 REPLXCVT DC     A(0)       ADDR OF XCVT IN PTN WHERE REPL IS BUILT
    REPLSIZ  EQU    *-REPL     SIZE OF REPL FOR INTERNAL USE
    *
    *  EQU'S FOR FLAG BYTE SUPFLAG
    *
    SUPFDEL  EQU    X'01'      EP DELETE OPTION
    SUPFDPTH EQU    X'02'      QPOS=DPATCH
    SUPFIRST EQU    X'04'      QPOS=FIRST
    SUPFRPTH EQU    X'08'      ECB REPATCH OPTION
    SUPFREEP EQU    X'10'      FREE=P
    SUPFPTNS EQU    X'20'      PTN=SLAVE
    SUPFPTNM EQU    X'40'      PTN=MASTER
    SUPNFREE EQU    X'80'      DO NOT FREE P OR WORK AREA (PTIME)
    *
```

Figure C-54. PROBL

```
    *  PROBLEM PARAMETER LIST FORMAT :
    *
000 PROBL    CSECT
000 PROBLNTH DC     H'4'       LENGTH OF PROBL INCLUDING THIS WORD  (MIN=4)
002          DC     X'0'       RESERVED FOR TASK MGMT
003 PROBID   DC     X'0'       ID VALUE (MAX=255)
004 PROBPARM DC     F'0'       PARAM VALUES (VARIABLE NUMBER)
```

Figure C-55 (1 Of 3). SCVT

```
000 SCVT      DSECT
    *
    **        COMMUNICATIONS VECTOR TABLE
    *
    *            - THE SCVT IS POINTED TO BY THE XCVT (XCVTCVTS)
    *            - THE SCVT CONTAINS SUBSYSTEM CONTROL INFORMATION
    *
000 SCVTDDSE DC     A(0)                  DDS ADDRESS OF EXTENSION LIST
004 SCVTXCVT DC     A(0)       POINTER BACK TO XCVT
008 SCVTLFLG DC     X'00'                 DATA BASE FLAGS
009          ORG    SCVTLFLG
008 SCVTLOG1 DC     A(0) .           DATA BASE LOG FREQUENCIES
00C SCVTLOG2 DC     A(0) .
010 SCVTLOG3 DC     A(0) .
014 SCVTTMCT DC     A(0) .           -> TASK MANAGEMENT CONTROL TABLE
018 SCVTTIME DC     A(0) .           -> TIME AND DATE IN DATA BASE
01C SCVTTQET DC     A(0) .           -> TIME MANAGEMENT TABLE
020 SCVTFLG1 DC     X'0'       FLAG BYTES
021 SCVTRSV  DC     AL3(0)     RESERVED
024 SCVTP1LO DC     A(0)       LO ADDR OF FIRST OR ONLY (THIS) PARTITION
028 SCVTP1HI DC     A(0)       HI ADDR OF FIRST OR ONLY (THIS) PARTITION
02C SCVTP2LO DC     A(0)       LO ADDR OF OTHER (MASTER OR SLAVE) PARTITION
030 SCVTP2HI DC     A(0)       HI ADDR OF OTHER (MASTER OR SLAVE) PARTITION
034 SCVTRWA  DC     A(0)                  A(RECORDER DCB)
038 SCVTPLST DC     A(0)       PTR TO LAST USED PSCB
03C SCVTDUMY DC     3F'0'      DUMMY PSCB
048          ORG    SCVTDUMY
03C SCVTDFCT DC     H'0'       DUMMY FREE COUNT
03E SCVTOID  DC     H'0'       PSCB ID
040 SCVTDFLG DC     X'0'       PSCBFLAG
041          ORG    SCVTDFLG
04C SCVTDNXT DC     A(0)       POINTER TO NEXT PSCB
044 SCVTDPRV DC     A(0)       POINTER TO PREVIOUS PSCB
048 SCVTGWLO DC     A(0)       LO ADDRESS OF GETWA CORE
04C SCVTGWHI DC     A(0)       HI ADDRESS OF GETWA CORE
050 SCVTT1BR DC     A(0) .     ADDRESS OF TYPE 1 SVC BRANCH TABLE
054 SCVTT2BR DC     A(0) .     ADDRESS OF TYPE 2 SVC BRANCH TABLE
058 SCVTMWA  DC     A(0) .     ADDRESS OF MSG HANDLER DCB
05C SCVTALOC DC     A(0)            DB    A(PRIMARY ARRAY LOCATOR TABLE)
060 SCVTAPBT DC     A(0)            DB    A(PAGE BOUNDARY TABLE)
064 SCVTDSTR DC     A(0)            DB    A(VS RESIDENT DATA BASE-START)
068 SCVTDEND DC     A(0)            DB    A(VS RESIDENT DATA BASE-END)
06C SCVTLKCB DC     A(0)                  LOCK CONTROL BLOCK CHAIN
070 SCVTALCB DC     A(0)                  A(DEFLOCK-LOCK CONTROL BLOCK)
074 SCVTUSRX DS     A     CHAIN OF MODULES REQUESTING STAE USER EXIT ROUTINES
078 SCVTSB00 DC     A(0)                  A(GETARRAY)
07C SCVTSB01 DC     A(0)                  A(PUTARRAY)
080 SCVTSB02 DC     A(0)                  A(GETITEM)
084 SCVTSB03 DC     A(0)                  A(PUTITEM)
088 SCVTSB04 DC     A(0)                  A(GETBLOCK)
08C SCVTSB05 DC     A(0)                  A(PUTBLOCK)
090 SCVTSB06 DC     A(0)                  A(MESSAGE HANDLER)
094 SCVTSB07 DC     A(0)                  A(DEFLOCK)
098 SCVTSB08 DC     A(0)                  A(LOCK)
09C SCVTSB09 DC     A(0)                  A(RECORD)
0A0 SCVTSB10 DC     A(0)       DDSOPEN    ENTRY POINT
0A4 SCVTSB11 DC     A(0)       DDSCLOSE   ENTRY POINT
```

Figure C-55 (3 Of 3). SCVT

```
 SCVTTTCB EQU    SCVTSB22              TIME TCB ADDRESS (DPPCTIME)
 SCVTSTAE EQU    SCVTSB23              A(DPPTSTAE)                 PRF#166
 SCVTDCHN EQU    SCVTSB24              CHANIN OF DMP/NDMP MODULES  PRF#166
 SCVTGWBS EQU    SCVTSB25              A(DPPTGWFW)
 SCVTSPWO EQU    SCVTSB26              A(PURGEWO)
 SCVTTWA  EQU    SCVTSB27              A(DPPTRGWA) - TRANSWA ROUTINE
 SCVTOTCB EQU    SCVTSB29              A(OTHER PTN'S  JS TCB)
 SCVTLNTH EQU    *-SCVT          LENGTH OF TABLE
 *               SCVTFLG1 DEFINITIONS
 *        NOTE THE SCVTFLG1 AND THE XCVT2PFG MUST BE THE SAME
 SCVTF2PT EQU    X'80'      TWO PARTITION OPERATION FLAG
 SCVTFMPT EQU    X'40'      MASTER PARTITION FLAG
 SCVTFSPT EQU    X'20'      SLAVE PARTITION FLAG
 *
 *        SCVTLFLG DEFINITIONS
 *
 SCVTRINT EQU    X'80'     DATA BASE REINITIALIZE (REFRESH) FLAG
 SCVTRDOP EQU    X'40'                 DA DATA BASE READ ONLY-TEMPORARILY
 SCVTRDOT EQU    X'20'                 DA DATA BASE READ ONLY-PERMANENT

HE DSECT NAMED 'SCVT     ' IS STOWED, CC=08
```

Figure C-56. STAEBLK

```
    ***                                                                ***
    *        STAEBLK                                                     *
    *                 DSECT USED TO DESCRIBE THE DUMP/NO DUMP CONTROL    *
    *                 BLOCKS. EACH BLOCK IS BUILT IN RESPONSE TO A'STAE' *
    *                 IMP COMMAND AND IS USED TO DESCRIBE THE OPTIONS IN *
    *                 EFFECT FOR A SPECIFIED LOAD MODULE                 *
    ***                                                                ***
000 STAEBLK  DSECT
000          DS     0D
000 STAENEXT DC     A(0)                    A(NEXT STAEBLK) OR ZERO
004 STAEABND DC     X'0'                    STAE/ABEND OPTION FLAGS
005 STAEMAXD DC     X'0'                    MAX.NO.OF DUMPS TO BE TAKEN
006 STAEDEND DC     X'0'                    NO.OF DUMPS TAKEN
007 STAELNAM DC     X'0'                    LENGTH OF LM NAME
008 STAENAME DC     CL8'                    MODULE NAME
010 STAEEND  DS     0D
    STAELNTH EQU    STAEEND-STAEBLK
    ***                                                                ***
    *        STAEABND - FLAG DESCRIPTION                                 *
    ***                                                                ***
    STAEDUMP EQU    X'80'                   A DUMP IS REQUESTED
    *               CORRESPONDS TO TCBCREQ (TCBDUMP) FLAGS IN TCBCMP FIELD
    STAESTEP EQU    X'40'                   A STEP ABEND HAS BEEN REQUESTED
    *               CORRESPONDS TO TCBCSTEP (TCBSTEP) FLAGS IN TCBCMP FIELD
```

Figure C-56.1. STAEXBK

```
000 STAEXBK  DSECT
000 STABKNXT DC     A(0)                ADDRESS OF NEXT BLOCK OR ZERO
004 STABKLEN DC     H'0'                SIZE OF THIS BLOCK
006 STABKCT  DC     H'0'                COUNT OFL M NAMES IN THIS BLOCK
008 STABKEPN DC     CL8' '              ENTRY POINT NAME OF EXIT ROUTINE
010 STABKEPA DC     A(0)                ADDRESS OF ROUTINE
014 STABKLMN DC     CL8' '              NAMES OF LOAD MODULES FOR WHICH THIS
    *                                   EXIT ROUTINE IS TO BE PASSED CONTROL
    *                                   IF THEY ABEND
    STABKMSZ EQU    1024                MAXIMUM SIZE OF ATAEXBK
```

Figure C-57. TCBX

```
000 TCBX      DSECT
    *
    ** TCBX = TCB EXTENSION - ADDRESSED BY TCBUSER FIELD OF TCB
    *
    *             - THE TCBX IS POINTED TO BY THE TCB (TCBUSER)
    *             - THE TCBX CONTAINS TASK RELATED INFORMATION
    *
000 TCBXNEXT DC     A(0) .     POINTER TO NEXT TCBX ON THIS CHAIN
004 TCBXNAME DC     CL8' ' .   TASK NAME IF INDEPENDENT - BLANKS OTHERWISE
00C TCBXLCB  DC     A(0) .     LCB CHAIN ORIGIN
010 TCBXWQ   DC     A(0) .     WQ CHAIN ORIGIN
014 TCBXCWQ  DC     A(0) .     CURRENT WQE IN PROCESS (ALREADY DECHAINED)
018 TCBXDWQ  DC     A(0) .     DPATCH WQ CHAIN
01C TCBXTGWA DS     2F         TASK (AT) GETWA CHAIN ORIGIN  (DUMMY GFBE)
024          ORG    TCBXTGWA
01C TCBXTFWD DC     A(0)         DUMMY GFBE FORWARD POINTER
020 TCBXTBKW DC     A(0)         DUMMY GFBE BACKWARD POINTER
024 TCBXQGWA DS     2F         WQ (AP) GETWA CHAIN ORIGIN (DUMMY GFBE)
02C          ORG    TCBXQGWA
024 TCBXQFWD DC     A(0)         DUMMY GFBE FORWARD POINTER
028 TCBXQBKW DC     A(0)         DUMMY GFBE BACKWARD POINTER
02C TCBXDCVT DC     A(0) .     POINTER TO DPPXCVT
030 TCBXRSTB DC     A(0) .     POINTER TO RESOURCE TABLE
034 TCBXPARM DC     A(0)       POINTER TO PROBLEM PARAMETERS
038          ORG    TCBXPARM
034 TCBXSMON DC     A(0)       CHAIN WORD FOR TMCTSMON CHAIN
038 TCBXTCB  DC     A(0)       POINTER TO TCB
03C TCBXECB  DC     A(0) .     ECB DPPTPMON WAITS ON FOR POST BY DPPTPSVC
040 TCBXLECB DC     A(0) .     ECB DPPTPMON WAITS ON FOR POST BY DPPTSMON
044 TCBXFLG1 DC     X'0' .     FLAG BYTE 1
045 TCBXFLG2 DC     X'0' .     FLAG BYTE 2
046 TCBXLQL  DC     X'0' .     LIMIT QUEUE LENGTH
047 TCBXCQL  DC     X'0' .     CURRENT QUEUE LENGTH
048 TCBXHWQL DC     X'0' .     HIGH WATER QUEUE LENGTH
049 TCBXFLG3 DC     X'0'       FLAG BYTE 3  QUEUE HOLDER/PROCESSOR FLAGS
04A TCBXPRTY DC     H'0' .     PRIORITY
04C TCBXCUWQ DC     A(0) .     CLEAN-UP WQ
050 TCBXPECB DC     A(0) .     ECB DPPTPMON WAITS ON FOR POST BY DPPTDLMP
    TCBXLNTH EQU    *-TCBX     LENGTH OF TCBX
    *  FLAG 1 DEFINITIONS
    TCBX1DOR EQU    X'04'      TASK DORMANT, WQ IS EMPTY
    TCBX1CHP EQU    X'08'      NEW TCBX REQUIRES CHAP
    TCBX1TCB EQU    X'10'      TCB NEEDED - USED BY DPPTSMON
    TCBX1LCB EQU    X'20'      RENT EP ADDRESS NEEDED - USED BY DPPTSMON
    TCBX1TRM EQU    X'40'      TASK TERMINATION REQUIRED
    TCBX1ACT EQU    X'80'      CURRENT WQ ACTIVE
    *  FLAG 2 DEFINITIONS
    TCBX2DPI EQU    X'01'      DPATCH - IMMEDIATE
    TCBX2DPU EQU    X'02'      DPATCH - UNCONDITIONAL
    TCBX2DPW EQU    X'04'      DPATCH - WHENEVER (NEXT TIME TASK IS DORMANT)
    TCBX2DPC EQU    X'08'      DPATCH - CONDITIONAL (IF TASK IS DORMANT NOW)
    TCBX2DP  EQU    TCBX2DPI+TCBX2DPU+TCBX2DPC+TCBX2DPW
    *  FLAG 3 DEFINITIONS
    TCBX3QH  EQU    X'80'      THIS IS A QUEUE HOLDER
    TCBX3QP  EQU    X'40'      THIS IS A QUEUE PROCESSOR
    TCBX3SEQ EQU    X'20'      QHOLDER IS DEFINED SEQUENTIAL
    TCBX3SEL EQU    X'10'      QHOLDER IS SEQUENTIAL AND SELECTED

    TCBX3HLD EQU    X'08'      QH OR QP IS HELD,DO NOT START NEW WORK
    TCBX3NOP EQU    X'04'      QH - DO NOT ACCEPT PATCHES
    *  QUEUE HOLDER/QUEUE PROCESSOR EXPANSION TO TCB EXTENSION.
    *         PRESENT ONLY IF FLAG 3 BIT TCBX3QH OR TCBX3QP IS ON
054          ORG    TCBXPECB+4
054 TCBXQCT  DC     H'0'       NUMBER OF ENTRIES IN ADDRESS TABLE
056 TCBXQUCT DC     H'0'       USE COUNT. NUMBER OF WQ'S PROCESSED BY QUEUE
    *                          NUMBER OF WQ'S REMOVED FROM QH
058 TCBXQCUR DC     F'0'       IF QH, ADDR OF LAST QP TO TAKE WORK
    *                          IF QP, ADDR OF QH WORK WAS TAKEN FROM LAS
05C          DC     F'0'               RESERVED
    TCBXQMAX EQU    21         MAXIMUM NO. OF CONNECTED BLOCKS(QP-QH XREF
060 TCBXQADR DC     A(0)       UP TO 21 ADDRESSES OF CONNECTED BLOCKS.
    *                          IF QH, ADDR OF QP'S THAT CAN TAKE WORK
    *                          IF QP, ADDR OF QH'S FROM WHICH WORK CAN BE
    *                          TAKEN
```

Figure C-58. TIMED

```
000 TIMEC     DSECT
    ***
    *         TIMF ARRAY DSECT
    ***
000 TIMEHS    DC      F'0'                 TOC IN 10 MIL UNITS
004 TIMETOC   DC      F'0'                 TCC IN DECIMAL 10 MIL UNITS-HHMMSSTH
008 TIMEJCAY  DC      F'0'                 JULIAN CATE-CCYYDCCC
00C TIMEMCAY  DC      F'0'                 CAY OF MONTH CATE-OMMDCYYC
010 TIMEEBC   CC      CL10' '              EPCCIC CATE- DD/MMM/YY
01A TIMEBCAY  DC      H'0'                 DAY CF YEAR - BINARY
01C TIMEECB DC        F'0'                 PTIME ECB
020 TIMECFAC  DC      D'0'                 CCNVERSICN FACTOR
028 TIMERCLK  DC      D'0'                 AREA TC STORE CLCCK
030 TIMELOCK  DC      A(0)                 A(TIME LOCK BLOCK)
034 TIMEINTL  DC      F'0'                 SYSGEN PTIME INTERVAL
038 TIMEECB2  DC      F'0'                 DELETE ECB
03C TIMEPRTY  DC      H'0'                 PTIME CISPATCHING PRIORITY
03E TIMEFREQ  DC      X'0'                 TIME FAILCVER FREQ
03F TIMEFUPD  DC      X'0'                 TIME FAILOVER UPDATE
040 TIMEENC CS                             OD
    TIMELNGH ECU                           TIMEENC-TIMED
```

Figure C-59. TMCT

```
000 TMCT     DSECT
    *
    **  TMCT = TASK MANAGEMENT CCNTROL TABLE - PCINTED TO BY THE SCVT
    *
    *             - THE TMCT IS PCINTED TC BY THE SCVT (SCVTTMCT)
    *             - THE TMCT CONTAINS TASK MANAGEMENT INFCRMATION
    *
000 TMCTAIND DC      A(0) .    CHAIN ORIGIN FOR INDEPENDENT TCBX'S
004 TMCTADEP DC      A(0) .    CHAIN ORIGIN FOR DEPENDENT TCBX'S
008 TMCTFREE DC      A(0) .    CHAIN ORIGIN FOR FREE TCBX'S
00C TMCTSMON DC      A(0)      CHAIN ORIGIN OF TCBX'S REQUIRING SMCN SERVICES
    *                                (CHAINEC BY TCBXSMCN WORD)
010 TMCTSECB DC      A(0) .    ECB DPPTSMON WAITS CN FCR POST BY PMCN CR SVC
014 TMCT#ACT DC      H'0' .    CURRENT NUMBER CF ACTIVE TCBX'S
016 TMCT#FRE DC      H'0' .    CURRENT NUMBER CF  FREE  TCBX'S
018 TMCT#TCB DC      H'0'      NUMBER OF TCBX'S GOTTEN AT INIT TIME
01A TMCT#HIX DC      H'0'      HIWATER NUMBER OF TCBX'S IN SYSTEM
01C TMCTFLG1 DC      X'0' .    FLAG BITS
01D TMCTRSV1 DC      X'0' .    RESERVED
01E TMCTLMP  DC      H'0' .    HIGHEST POSSIBLE LIMIT PRTY FOR ANY DPPTFMON
    TMCTLMPD EQU     3 .       DIFFERENCE BETWEEN DPPTSMCN'S LMP AND TMCTLMP
020 TMCTLCBA DC      A(0) .    CHAIN ORIGIN FOR TMCT-LCB CHAIN
024 TMCTCCVT DC      A(0)      ADDRESS OF DPPXCVT
028 TMCTTMCB DC      A(0) .    POINTER TC TIME CCNTRCL BLOCK
02C TMCTEXGW DS      2F        EXCLUSIVE (PC) GETWA CHAIN CRIGIN (DUMMY GFBE)
034          ORG     TMCTEXGW
02C TMCTEFWD DC      A(0)        DUMMY GFBE FORWARD PCINTER
030 TMCTEBKW DC      A(0)        DUMMY GFBE BACKWARD PCINTER
034 TMCTETXR DC      A(0) .    ADDRESS OF ETXR USED WITH ATTACH OF PMCN
038 TMCTCFMB DC      A(0)      ADDRESS OF FIRST GETWA/FREEWA MAIN BLOCK
03C          ORG     *-4
038 TMCT#GSZ DC      X'00'     NUMBER OF GETWA SIZES
039          ORG     *+3
03C TMCTCFCB DC      A(0)      ADDRESS OF FIRST GFCB CN CHAIN
040 TMCTREPL DC      A(0)      CHAIN ORIGIN FOR REPATCH LIST'S
044 TMCTPECB DC      A(0) .    ECB DPPTCLMP WAITS CN FCR PCST BY DPPTSMON
    TMCTLNTH EQU     *-TMCT    LENGTH OF TMCT - SUBSYSTEM AND GETWA TABLES
    *  TMCTFLG1 DEFINITIONS
    TMCTLCBD EQU     X'80'     REQUEST FOR LCB-DELETE PROCESSING BY DPPTSMON
    TMCTFLMP EQU     X'40'     INDICATES ONE TMCT-LCB HAD LCBFLMP + LCBFDEL
000 GFMB     DSECT
    *
    ** GETWA/FREEWA MAIN BLOCK - CNE FOR EACH POSSIBLE GETWA SIZE
    **                         - WILL BE APPENDED CN END OF TMCT
    *                          - CHAIN PCINTEC TO BY TMCT (TMCTGFMB)
    *
000 GFMBFCNT DC      H'0'      CURRENT NUMBER CF FREE BLCCKS
002 GFMB#BLK DC      H'0'      INITIAL NUMBER OF BLOCKS ALLOCATED
004 GFMBSIZE DC      F'0'      SIZE OF BLOCKS IN THIS POOL
008 GFMBID   DC      X'00'     ID FIELD TO ASSCCIATE GFBE'S TC A GFMB
009          ORG     GFMBID
008 GFMBGFCB DC      A(0) ADDRESS CF GFCB FOR THIS SIZE BLOCK
    GFMBLNTH EQU     *-GFMB    LENGTH OF GFMB
    *
```

Figure C-60. WQE

```
000 WQE       DSECT
    *
    ** WCE = WORK QUEUE ELEMENT - CHAINED TO TCBX
    *
000 WQNEXT    CC     A(0) .     PCINTER TO NEXT WQE IN CHAIN
004 WQLCB     DC     A(0) .     POINTER TO LCB FCR ENTRY POINT
008 WOTCBX    DC     A(0) .     FCINTER TO TCBX WHERE WQ IS CHAINED TO
00C WQFLAGS   DC     X'0' .     FLAG BITS
00D WQFPATCH  DC     X'0'       FLAGS FROM PATCH PARAMETER LIST
00E WQFRESV   DC     X'00'      RESERVED
00F WCIC      DC     X'00'      PRCBLEM PARAMETER IC VALUE
010 WQECBAD   DC     A(0)       ADDRESS OF ECB SPECIFIED IN ECB= OPERAND
014 WCPTCB    DC     A(0) .     PATCHING TASK'S TCB ACDRESS
018 WQFREELN  DC     A(0) .     FREEMAIN= OPERAND LENGTH
01C WQFREEAD  DC     A(0) .     FREEMAIN= OPERANC ADDRESS
020 WQECBCOD  CC     A(0) .     COMPLETICN TO POST USER'S ECB WITH
024 WCPARAM   DC     A(0) .     PCINTER TO PROBLEM PARAMETERS
028           ORG    WQPARAM
024 WQPROBL   CC     XL8'0'     PROBL WILL BE MCVED INTO WQE IF LE 8 BYTES
02C           ORG    WQPRCBL
024 WCPRCBLN  DC     H'0'       LENGTH OF PRCBL INCLUDING THIS WCRD  (MIN=4)
026           DC     X'0'       RESERVED FOR TASK MGMT USE
027 WQPRCBID  CC     X'0'       IC-VALUE (MAX=255)
028 WQPROBPA  DC     F'0'       PARAMETER VALUE
    WQLNTH    EQU    *-WQE      LENGTH OF WQE
    WQ        EQU    WQE
    *  WQFLAGS EQU'S
    WQFLFREP  EQU    X'01'      FREE=P WAS SPECIFIED
    WQFLPARM  EQU    X'04'      PROBL PARMLIST MOVED INTO WQE
    WQFLABND  EQU    X'08'      ABEND OCCURRED WHILE PROCESSING THIS WQE
```

Figure C-61. XCVT

```
000 XCVT      DSECT
    *
    **  DPPXCVT = COMMUNICATIONS VECTOR TABLE
    *
    *            - THE XCVT IS POINTED TO BY EACH TCBX (TCBXDCVT)
    *            - THE XCVT CONTAINS CONTROL INFORMATION FOR SRTOS
    *
000 XCVTRESV DC     A(0) .    RESERVED WORD - MUST BE ZERO
004 XCVTSVC1 DC     A(0) .    TYPE 1 SVC INSTRUCTION
008 XCVTSVC2 DC     A(0) .    TYPE 2 SVC INSTRUCTION
00C XCVTSVC4 DC     A(0) .    TYPE 4 SVC INSTRUCTION
010 XCVTPGSZ DC     A(0)      PAGE SIZE
014 XCVTSBDT DC     CL8' '              DATE OF LAST SYSTEM BUILD
01C XCVTSBOP DC     A(0)      HI-ORDER BYTE = SYSTEM BUILD OPTION FLAGS
    *                         3 LO-ORDER BYTES = RESERVED - FAILOVER/RESTART
020 XCVT2PFG DS     X                   2 PARTITION FLAGS
021          ORG    XCVT2PFG
020 XCVTCVTS DC     A(0)      -> DPPXCVTS          CVT
024 XCVTSSV1 DC     A(0) .    SUBSYSTEM VECTOR TABLE POINTERS
028 XCVTSSV2 DC     A(0)        DISPLAY MGMT CVT POINTER
02C XCVTSSV3 DC     A(0)        ENERGY MGMT CVT POINTER
030 XCVTSSV4 DC     A(0)        DATA ACQ CVT POINTER
034 XCVTSSV5 DC     A(0) .
038 XCVTSSV6 DC     A(0) .
03C XCVT2PTX DC     A(0)      ADDR OF THE OTHER PARTN XCVT - TWO PARTN OPERN
040 XCVTPFRF DC     A(0)      ADDRESS OF THE PAGE FREE (UNFIX) ROUTINE
    XCVTDPN  EQU    XCVTSSV4
    XCVTECVT EQU    XCVTSSV3
    XCVTDCVT EQU    XCVTSSV2
    *              SYSTEM BUILD OPTION FLAGS
    XCVTFDDS EQU    X'01'     DUPLICATE DATA SET SUPPORT FLAG
    XCVTFFAL EQU    X'02'     FAILOVER RESTART FLAG
    XCVTFEXT EQU    X'04'     EXTERNAL INTERRUPT HANDLER FLAG
    XCVTINTF EQU    X'08'     END OF INITIALIZATION FLAG
    XCVTPRS  EQU    X'10'     PRE-RESTART FLAG
    XCVTIPL  EQU    X'20'     INITIAL IPL FLAG
    XCVTCPU  EQU    X'40'     SAME CPU AS IPL FLAG
    XCVTPROB EQU    X'80'               PRE-PROBE FLAG
    *               XCVT2PFG DEFINITIONS
    *        NOTE THE  XCVT2PFG AND THE SCVTFLG1 MUST BE THE SAME
    XCVTF2PT EQU    X'80'               TWO PARTITION OPERATION FLAG
    XCVTFMPT EQU    X'40'               MASTER PARTITION
    XCVTFSPT EQU    X'20'               SLAVE PARTITION
    XCVTRSNC EQU    X'10'               SLAVE PTN HAS BEEN RE-SYNC'ED
    XCVTLNTH EQU    *-XCVT         LENGTH OF TABLE
```

Appendix D. INTERNAL MACROS

This section contains a list of internal macros and their calling sequences. These macros are restricted to use by the Special Real Time Operating System.

CBFREE

The CBFREE macro releases control of a work area in protected storage. This area must have been previously obtained by executing a CBGET macro call.

Symbol	CBFREE	$\text{ADDR}=\begin{Bmatrix}\text{(r)}\\ \text{address}\end{Bmatrix}$ $\left[\begin{Bmatrix}\text{, DCVTR}=\text{(r)}\\ \text{,DCVTLOC}=\begin{Bmatrix}\text{(r)}\\ \text{address}\end{Bmatrix}\end{Bmatrix}\right]$
Where	'r' is a general-purpose register, 2-12	

ADDR=
: Indicates the address of the protected storage area to be freed. If 'r' is specified, the register contains the address of the work area as returns to the caller after a CBGET macro execution. If an address is specified, it is the label of a fullword that contains the address of the work area as returned to the caller after a CBGET macro execution.

DCVTR=r
: Where 'r' is the general-purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=r
: Where 'r' is the general-purpose register (2-12) enclosed in parentheses having the address of a 4-byte core location that contains the address of the XCVT.

DCVTLOC=address
: Where 'address' is the label of a 4-byte core location that contains the address of the XCVT.

CBGET

The CBGET macro is used to obtain a protected storage area to be used as a work area by the Special Real Time Operating Systems routines. The address of the storage area is returned in register 1.

Symbol	CBGET	$\left\{\begin{array}{c}\text{(r)}\\ \text{length}\end{array}\right\}\left[\left\{\begin{array}{l},\ \text{DCVTR=(r)}\\ ,\text{DCVTLOC=}\left\{\begin{array}{c}\text{(r)}\\ \text{address}\end{array}\right\}\end{array}\right\}\right]$
Where	'r' is a general-purpose register, 2-12	

length=
: is the length of the requested work area that can be specified in any RX-type format or in a general-purpose register.

DCVTR=r
: Where 'r' is the general-purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
: Where 'r' is the general-purpose register (2-12) enclosed in parentheses having the address of a 4-byte core location that contains the address of the XCVT.

DCVTLOC=address
: Where 'address' is the label of a 4-byte core location that contains the address of the XCVT.

DPPFIX

The DPPFIX macro provides the facility for fixing pages in a virtual storage environment.

```
 Symbol          DPPFIX               ⎧  (r)    ⎫
                               HGHADDR=⎩address ⎭

                                      ⎧  (r)    ⎫
                               LOWADDR=⎩address ⎭
                            ⎡⎧ ,DCVTR=(r)         ⎫⎤
                            ⎢⎨          ⎧  (r)    ⎫⎬⎥
                            ⎣⎩ ,DVTLOC=⎩address ⎭⎭⎦
            Where 'r' is a general-purpose register, 2-12
```

HGHADDR=

is the address of the upper boundary of a virtual storage area that is to be page fixed. This address will be rounded up to the next page boundary, if required. If 'r' is specified, the register contains the upper boundary address. If an address is specified, it is a label of a fullword that contains the address of the upper boundary.

LOWADDR=

is the address of the lower boundary of a virtual storage area that is to be page fixed. This address will be rounded down to the previous page boundary, if required. If 'r' is specified the register contains the lower boundary address. If an address is specified, it is a label of a fullword that contains the address of the lower boundary.

DCVTR=r

Where 'r' is the general-purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)

Where 'r' is the general-purpose register (2-12) enclosed in parentheses having the address of a 4-byte core location that contains the address of the XCVT.

DCVTLOC=address

Where 'address' is the label of a 4-byte core location that contains the address of the XCVT.

DPPFREE

The DPPFREE macro provides the facility for freeing pages in a virtual storage environment that have been page fixed as the result of a DPPFIX macro call.

```
 Symbol        DPPFREE                    { (r)     }
                                HGHADDR= { address },

                                          { (r)     }
                                LOWADDR= { address }
                          [ { ,DCVTR= (r)               } ]
                          [ {             { (r)     }   } ]
                          [ { ,DVTLOC= { address }   } ]
        Where 'r' is a general-purpose register, 2-12
```

HGHADDR=
is the address of the upper boundary of a virtual storage area that is to be page freed. This address will be rounded up to the next page boundary, if required. If 'r' is specified, the register contains the upper boundary address. If an address is specified, it is a label of a fullword that contains the address of the upper boundary.

LOWADDR=
is the address of the lower boundary of a virtual storage area that is to be page freed. This address will be rounded down to the previous page boundary, if required. If 'r' is specified, the register contains the lower boundary address. If an 'address' is specified, it is a label of a fullword that contains the address of the lower boundary.

DCVTR=r
Where 'r' is the general-purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(5)
Where 'r' is the general-purpose register (2-12) enclosed in parentheses having the address of a 4-byte core location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte core location that contains the address of the XCVT.

SETPSW

The SETPSW macro provides the Special Real Time Operating System with the capability of changing the current program status word (PSW) to alter the storage protect key, mode, and/or system mask. On return, register zero will contain a parameter that will enable the user to restore the PSW the previous status (i.e., status immediately prior to the execution of the SETPSW macro)

Symbol	SETPSW	$\left\{\begin{array}{l} \left[\text{KEY}=\begin{Bmatrix}0\\ \text{TCB}\end{Bmatrix}\right] \quad \left[,\text{STATE}=\begin{Bmatrix}\text{S}\\ \text{PP}\end{Bmatrix}\right] \quad \left[,\text{INT}=\begin{Bmatrix}\text{D}\\ \text{E}\end{Bmatrix}\right] \\ \text{REG}=(\text{r}) \end{array}\right\}$ $\left[\begin{Bmatrix},\text{DCVTR}=(\text{r})\\ ,\text{DCVTLOC}=\begin{Bmatrix}(\text{r})\\ \text{address}\end{Bmatrix}\end{Bmatrix}\right]$
	Where 'r'	is a general-purpose register, 2-12

KEY=
is used to set the storage key in the PSW. '0' provides the user with the supervisor storage key and 'TCB' sets the protect key to the task protect key.

STATE=
is used to set the mode in the PSW. 'S' puts the CPU in a supervisor mode and 'PP' puts the CPU in a problem program mode.

INT=
is used to set the system mask in the PSW. 'D' disables all I/O and external interrupts. 'E' enables all I/O and external interrupts.

REG=
is used to restore the PSW to the previous status. The register specified must contain the parameter that is returned in register zero as the result of a previous SETPSW macro call. The 'REG=' parameter and either the 'KEY=', 'STATE=', or 'INT=' parameters are mutually exclusive.

DCVTR=r
Where 'r' is the general-purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where 'r' is the general-purpose register (2-12) enclosed in parentheses having the address of a 4-byte core location that contains the address of the XCVT.

DCVTLOC=address

Where 'address' is the label of a 4-byte core location that contains the address of the XCVT.

WTFAILDS

The WTFAILDS macro writes the failover/restart data set.

```
Symbol          WTFAILDS       [{      ,DCVTR=(r)          }]
                               [{                          }]
                               [{               { (r)     }}]
                               [{  ,DCVTLOC=    { address }}]

          Where 'r' is a general-purpose register, 2-12
```

DCVTR=r
Where 'r' is the general-purpose register (2-12) that contains the address of the XCVT.

DCVTLOC=(r)
Where 'r' is the general-purpose register (2-12) enclosed in parentheses having the address of a 4-byte core location that contains the address of the XCVT.

DCVTLOC=address
Where 'address' is the label of a 4-byte core location that contains the address of the XCVT.

READER'S COMMENT FORM

IBM System/370 Special Real Time Operating System LY20-2228-1
Programming RPQ Z06751
Systems Logic Manual

Please comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you wish a reply, be sure to include your name and address.

LICENSED MATERIAL – PROPERTY OF IBM

COMMENTS

fold

fold

fold

fold

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A. FOLD ON TWO LINES, STAPLE AND MAIL.

Your comments, please . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold Fold

LICENSED MATERIAL – PROPERTY OF IBM

Fold Fold

IBM®

IBM System/370 Special Real Time Operating System Programming RPQ Z06751 Systems Logic Manual Printed in U.S.A. LY20-2228-1